BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2019 Bc. Martin Tilgner

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

PEDESTRIANS DETECTION IN TRAFFIC ENVIRONMENT BY MACHINE LEARNING

DETEKCE CHODCŮ VE SNÍMKU POMOCÍ METOD STROJOVÉHO UČENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Martin Tilgner**
AUTOR PRÁCE

SUPERVISOR — **Ing. Karel Horák, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2019

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Cybernetics, Control and Measurements**
Department of Control and Instrumentation

Student: Bc. Martin Tilgner ***ID:*** 174416
Year of study: 2 ***Academic year:*** 2018/19

TITLE OF THESIS:

Pedestrians Detection in Traffic Environment by Machine Learning

INSTRUCTION:

Pedestrians localisation and classification tasks in traffic environment are usually solved by convolutional neural networks. Such approach is used in this thesis. Objectives:

1. to make a bibliographic recherche about pedestrian detection methods from an autonomous car point of view
2. to make an own or to take over an existing dataset (training and verification part)
3. to make an implementation of selected or own machine learning algorithm
4. to make a code verification on CityPersons dataset and to evaluate algorithm's properties

REFERENCE:

1. Goodfellow I., Bengio Y., Courville A.: Deep Learning. MIT Press, 2016. ISBN 9780262035613. (deeplearningbook.org)
2. Buduma, N., Locascio, N. Fundamentals of Deep Learning. O'Reilly Media, Inc. 2017. ISBN 9781491925614.

Assigment deadline: 4. 2. 2019 ***Submission deadline:*** 13. 5. 2019

Head of thesis: Ing. Karel Horák, Ph.D.

doc. Ing. Václav Jirsík, CSc.
Subject Council chairman

ABSTRACT

This thesis deals with pedestrian detection using convolutional neural networks from the perspective of autonomous vehicle. Especially by testing these networks in the sense of finding a suitable practice of creating a dataset for machine learning models. A total of ten machine learning models of meta architectures Faster R-CNN with ResNet 101 as a feature extractor and SSDLite with the MobileNet_v2 feature extractor were trained. These models were trained on datasets of various sizes. The best results were achieved on a dataset of 5 000 images. In addition to these models, a new dataset aimed at pedestrians at night was created. Furthermore, a Python library was created for work with datasets and script for automatic creation of dataset.

KEYWORDS

Machine learning, Object detection, Pedestrian detection, Tensorflow, Faster R-CNN, SSDLite, Dataset, Small Night Pedestrian Dataset

ABSTRAKT

Tato práce se zabývá detekcí chodců pomocí konvolučních neuronových sítí z pohledu autonomního vozidla. A to zejména jejich otestováním ve smyslu nalezení vhodné praxe tvorby datasetu pro machine learning modely. V práci bylo natrénováno celkem deset machine learning modelů meta architektur Faster R-CNN s ResNet 101 jako feature extraktorem a SSDLite s feature extraktorem MobileNet_v2. Tyto modely byly natrénovány na datasetech o různých velikostech. Nejlépší výsledky byly dosaženy na datasetu o velikosti 5000 snímků. Kromě těchto modelů byl vytvořen nový dataset zaměřující se na chodce v noci. Dále byla vytvořena knihovna Python funkcí pro práci s datasety a automatickou tvorbu datasetu.

KLÍČOVÁ SLOVA

Machine learning, Detekce objektů, Detekce chodců, Tensorflow, Faster R-CNN, SSDLite, Dataset, Small Night Pedestrian Dataset

TILGNER, Martin. *Person Detection in Image by Machine Learning*. Brno, 2019, 92 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation. Advised by Ing. Karel Horák, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Tato práce se zabývá detekcí chodců v dopravním prostředí pomocí technik machine learning, konkrétně pomocí konvolučních neuronových sítí. Detekce osob, respektive chodců je zde prováděna z pohledu autonomního vozidla. Použití technik machine learning je pro detekci chodců a osob obecně v takovémto případě výhodné, protože se zde jedná o složitý problém, protože se mění jak prostředí, tak i objekty zájmu. Zároveň se jedná o velice odlišný problém, než na jaký jsou běžné machine learning modely stavěné, neboť zde nemůžeme udělat jednoduchý kompromis mezi přesností a rychlostí, protože i jedna False Positive detekce by mohla mít tragické následky.

Před vlastním řešením daného problému bylo nutné provést rešerši literatury věnující se tomuto tématu. První část této práce se proto věnuje principu fungování konvolučních neuronových sítí od úplných základů, přes klasifikaci až po detekci objektů. Vytváří tak teoretický základ celé práce. V druhé kapitole, která staví na poznatcích uvedených v první kapitole, jsou rozebrány aktuální State-of-the-Art řešení od konvolučních neuronových sítí pouze pro klasifikaci po meta-architektury pro detekci objektů využívající předchozí zmíněné sítě jako feature extraktory. V této kapitole jsou také porovnány výsledky jednotlivých sítí jak z hlediska rychlosti, tak z hlediska přesnosti. Jsou zde také zmíněny práce, zabývající se detekcí chodců pomocí těchto architektur.

Protože se u všech State-of-the-Art metod jedná o učení s učitelem, další kapitola se věnuje datasetům vhodným pro detekci chodců. Kvalita datasetu použitého při učení přímo ovlivňuje kvalitu výsledného modelu, a proto je důležité věnovat datasetu velkou pozornost. V této kapitole jsou popsány veřejně dostupné datasety. Dále se kapitola zabývá možnostmi tvorby datasetu pro robustnější modely jako je metoda Boostrap a metoda Cross Validace. Velká pozornost je věnována zejména datasetům, které byly nějakým způsobem využity v této práci. I přes velké množství dostupných datasetů bylo potřeba vytvořit i dataset vlastní. Vzhledem k absenci datasetu obsahující noční snímky chodců bylo rozhodnuto pro vytvoření nového datasetu zaměřeného právě na chodce v dopravním prostředí za špatných světelných podmínek – tedy brzy ráno, večer a v noci. Během tvorby tohoto datasetu bylo dbáno na obsažení co nejrozmanitějšího prostředí od centra města přes vesnice po horské silnice. Zároveň byl dataset pořízen za různých povětrnostních podmínek. Tento dataset byl poté také využit spolu s vybranými datasety z této kapitoly k vlastnímu experimentu.

Na základě rešerše odborné literatury zabývající se tímto tématem jsme vybrali dvě meta-architektury pro bližší testování, a to Faster R-CNN s ResNet 101 jako feature extraktorem a SSDLite s MobileNet_v2 jako feature extraktorem. Architektura Faster R-CNN byla vybrána zejména pro svou přesnost, zatímco SSDLite byla vybrána hlavně pro svou rychlost, a protože je optimalizována pro mobilní zařízení.

Právě tato vlastnost z ní činní ideálního kandidáta pro autonomní vozidla. Architektura Faster R-CNN byla přetrénována na datasetu KITTI, zatímco SSDLite byla předtrénována na datasetu COCO. Jako platformu pro vytváření machine learning modelů jsme se rozhodli použít TensorFlow a knihovnu TensorFlow Object Detection API.

Implementace vybraných meta-architektur je zájmem kapitoly číslo čtyři. Kromě implementace se zde také zabýváme vhodným poskládáním trénovacího datasetu z různých veřejně dostupných datasetů. Rozhodli jsme se pro použití metody Hold-Out s trénovacím, testovacím a validačním dataset. Velikost námi použitého trénovacího datasetu je 5000 snímků. Skládá se z datasetů KITTI, CityPersons, Pascal VOC a námi vytvořeného Small Night Pedestrian Dataset. Pro tvorbu a práci s datasetem jsme vytvořili Python knihovnu funkcí, která je také objektem zájmu této kapitoly. Jelikož ruční anotace datasetu je velice časově náročná, rozhodli jsme se vytvořit nástroj, který pomocí machine learning modelů je schopen automaticky anotovat snímky a tím ulehčit práci při tvorbě nových datasetů. V neposlední řadě se kapitola zabývá vytvořenými skripty pro detekci, ať už ze statického obrazu nebo videa.

Samotný experiment pak spočívá v hledání pseudo-optimálního nastavení při trénování těchto machine learning modelů. Nejprve jsme natrénovali obě meta-architektury na snímcích pouze z KITTI datasetu, poté na snímcích z datasetu CityPersons, KITTI, Small Night Pedestrian Dataset a Pascal VOC datasetu. Dále jsme měnili množství trénovacích snímků u datasetu na hodnoty 10, 100, 800, 2000 a 5000, ale již jen pro architekturu Faster R-CNN. V posledním kroku jsme vyzkoušeli změnu rozlišení feature extraktoru. Nejlepší nastavení jsme poté aplikovali i na architekturu SSDLite, abychom viděli, zda dojde i u jiné meta-architektury ke zlepšení. Testovací dataset byl pro všechna učení stejná. Skládal se ze snímků CityPersons, KITTI a Small Night Pedestrain Dataset.

Poslední kapitola této práce se věnuje rozboru výsledků. Evaluace přesnosti natrénovaných modelů byla vyhodnocena pomocí několika metrik. Používali jsme COCO metriku přímo z Object Detection API, dále jsme k vyhodnocení použili Pascal metriku a pro bližší rozbor výsledků jsme vytvořili skript, který kromě běžně používaných hodnot jako mAP a senzitivita (recall) zobrazí celou matici záměn (confusion matrix) a vypočte zní potřebné parametry. U skriptu je možné nastavovat různé parametry, jak má model vyhodnocovat. My jsme se rozhodli pro hodnoty IoU Threshold 50 % a hodnotu Score Threshold 70 %. Dalším z parametrů, který jsme sledovali je rychlost detekce. Rychlost byla vyhodnocena na dvou různých strojích běžících pod operačním systémem Windows 10. První z přístrojů – běžný notebook – používal low-end grafickou kartu NVIDIA MX 150, druhý z počítačů používal grafickou kartu NVIDIA Titan X.

Potvrdili jsme, že Faster R-CNN je pomalejší, ale přesnější. Zatímco SSDLite má výrazně nižší přesnost. Dosahuje však vysoké rychlosti i na slabších zařízeních. Rychlost Faster R-CNN dosáhla na NVIDIA MX 150 pouze 1.51 FPS. Na NVIDIA Titan X bylo dosaženo rychlosti 7.52 FPS. Rychlost SSDLite dosáhla 45.34 FPS na NVIDIA MX 150 a 47.31 FPS na NVIDIA Titan X. Je tedy patrné, že architekura Faster R-CNN není optimalizovaná pro mobilní zařízení.

Velkým problémem u SSDLite je chybná detekce malých objektů. Zde se hodnota mAP dosáhla maximálně 3.08 % to je přibližně desetkrát méně než nejlepší výsledek Faster R-CNN (34.89 %). Faster R-CNN dosáhla bez rozšíření rozlišení feature extraktoru mAP na malých objektech 29.81 %. Problém s detekcí malých objektů lze tedy řešit právě změnou rozlišení feature extraktoru. Rozšířením trénovacího datasetu se zvyšovala přesnost logaritmicky u SSDLite a lineárně u architektury Faster R-CNN. Předpokládaná závislost je logaritmická. Lineární závislost u Faster R-CNN se dá vysvětlit pomocí transfer learningu. Síť Faster R-CNN byla předtrénována na KITTI datasetu, zatímco SSDLite byla předtrénována na COCO datasetu. Síť Faster R-CNN tedy již překročila silně nelineární část logaritmického grafu a při dalším učení tedy je závislost přibližně lineární.

Velký vliv měla také rozmanitost datasetu. Zejména u SSDLite došlo k výraznému snížení False Positive detekcí. Zásadní vliv má pak i dataset použitý při trénování, u sítě Faster R-CNN nejlepší výsledek mAP na KITTI datasetu byl 99.23 %, nicméně při evaluaci na ostatních datasetech bylo dosaženo horších výsledků. Nejlepšího dosaženého výsledku mAP u SSDLite bylo dosaženo na Small Night Pedestrian Dataset a to 72.09 %.

DECLARATION

I declare that I have written the Master's Thesis titled "Person Detection in Image by Machine Learning" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to express my very great appreciation to Ing. Karel Horák, Ph.D. for professional guidance, consultation, patience and suggestions and comments on my work. Furthermore, I would like to thank my family for all their help, motivation and support during my studies.

Brno ....

author's signature

Contents

List of Figures

List of Tables

INTRODUCTION

Artificial Intelligence (AI) is all around us, even though most people do not realize it at all. For example, we can find it in the search engines like Google, in the smart cameras where it can be used for focusing and mode selection e.g., in the mobile devices where it can be used for power consumption reduction, but it can also be found in cars or in the financial sector. We can find its applications also in the medicine and pharmacy in medical images analysis or in the drugs development. In last few years, AI has achieved a great deal of growth, due to greater media interest, but mainly due to the significant increase in computing power and the optimization of GPUs for AI needs. Last but not least the BigData processing demand is growing.

In the future, we can expect further development in this field, especially in social areas, robotics, autonomous transport and the use of AI in military technologies. AI brings a lot of benefits to humanity. On the other hand, it can be a danger about which leading figures of the world of science like Stephen Hawking or Elon Musk warned. Thanks to the development of AI and hardware we are able to create applications that were unthinkable a few years ago.

The person detection is useful in lots of applications like smart cities and autonomous transport. For example, the self-driving cars need to detect objects like persons and others to drive safely. Security is one of the main motivations for self-driving car implementation. Another example is the services personalization. Metrics like gender, age, hobbies, friends list or shopping habits can be used to understanding better customer type which leads to improving customers experience and higher profit for the owner.

Due to reasons mentioned above, a number of literature deals with this problem. For example, the author of this paper [1] deals with pedestrian detection in subways. It is environment characteristic with a large traffic flow and high pedestrian occlusion. Papers [2], [3], [4] confirmed that deep convolutional neural networks outperform a traditional approach to person detection task. Some other papers like [2], [3], [4] focus on comparing different architectures.

Due to focusing on using machine learning algorithm for pedestrian detection in autonomous car driving systems, it is good to mention the work [7] which is dealing with deep reinforcement learning for real-world autonomous driving systems. The authors introduce some of existing solutions and real world's problem of autonomous driving systems. A different solution for autonomous driving task is introduced in the work [8]. They suppose third paradigm for vision-based autonomous driving system - map an input image to a small number of key perception indicators that directly relate to the affordance of a road / traffic state for driving. Work [9] focus on

predicting the future of autonomous vehicle's surrounding which is necessary for safe and efficient transport. For the prediction role, the authors used a deep learning-based method. Another example of work related to this field of research is [10]. The authors aim at convolutional neural network for object detection. While most of the research focus on accuracy or speed, this work deals with object detection for autonomous driving, so the authors emphasize that for real use, the resulting system must meet both accuracy and speed requirements, as well as small model size and energy saving.

This work deals with person detection using convolutional neural network. Convolutional neural networks (CNN) have been around since the 90s. However, they were not able to competitively compare to other detection solutions. But these days CNN shows their potential and outperforms other object detectors. The reason for that is simple - nowadays we have got much more computational power and lots of data.

This work is divided into several logical parts. The first part describes the principle of convolutional neural networks. The next chapter deals with State-of-the-Art detectors using convolutional neural networks and their comparison in terms of speed and accuracy. The following chapter deals with public data sets and data pre-processing and an estimation of the precision of the resulting model. The fourth chapter focus on experiment design. This section discusses the design of the experiment and its implementation, including a step-by-step guide to the experiment. The last chapter is about experiment results. The behavior of individual machine learning models is described here.

1 CONVOLUTIONAL NEURAL NETWORK

Convolutional neural network originated as a modification of the deep neural network. They are primary used in image recognition because they need minimal image pre-processing from their principle. They are also used in natural language processing. This chapter will in short explain, how does CNN work.

1.1 Neuron and Neural network

Neuron is a basic computation unit of neural network. It consists of two parts. The first part is basically only a matrix multiplication, which performs linear function shown in the equation (1.1).

$$WX + b = y \tag{1.1}$$

Where the W is a vector of weights, the X is a vector of inputs and the b is a vector of biases. The y is output of this linear function.

The second part of neuron is nonlinear. It is usually called activation function. The output of this function is a final output of this neuron, it is also called score.

Usually used activation functions are Sigmoid (1.2), Tanh (1.3), ReLU (1.4) and Leaky ReLU (1.5).

$$S(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} + e^{-x}}{e^{x} - e^{-x}} \tag{1.3}$$

$$f_{ReLU}(x) = max(0, x) \tag{1.4}$$

$$f_{lReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{else} \end{cases} \tag{1.5}$$

Simple neuron can be used as a linear classifier. But many neurons are used in much larger neural networks (NN) by connecting their outputs as an input to another neurons. This system of connecting creates a layer of neurons. The topology of the network is then determined by the interconnection of the individual layers. The output of neurons in the last layer of this neural network is the output of this whole network. Typically, such a network consists of an input layer, several hidden layers, and an output layer. The architecture of NN can be called a model. Training of this model is based on setting the weights.

1.2 SoftMax function

For input classification by neural network we need to turn output scores of neurons in the last layer into probabilities for each classified class. To achieve this, the SoftMax function (1.6) is used. It takes an un-normalized vector y (of scores) and normalizes it into a probability distribution.

$$S(x) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{1.6}$$

1.3 Loss function

As mentioned before, training of the model is based on setting the weights and biases the way that the output score is high for correct class and small for incorrect class. We can measure output scores like the distance of the score from training label for entire training dataset and then average them. This is called training loss, it is also the average cross-entropy over the training dataset. The smaller training loss is, the better is the model learned. Learning is then setting the weights of the model to minimalize the cross-entropy commonly based on the cross-entropy gradient descending.

1.4 Convolutional neural network

Convolutional Neural Network is neural network, which is usually used for image classification or detection. So basic neural network has an input values in the form of a vector. In CNN a volume (most commonly a multichannel image) serves as an input. The main difference between NN and CNN is that CNN uses convolution operation instead of matrix multiplication in at least one of its layers. The general idea of CNN is forming a pyramid of convolutional and pooling layers, which decrease width and height of the image, while increasing its depth. At the end of this pyramid is located a classifier, which is usually a fully connected neural network. So, the first part of this pyramid structure – convolutional and pooling layers – is the feature extractor. The second part – fully connected layers – is used for classifying. Pyramid structure is displayed in the figure 1.4.

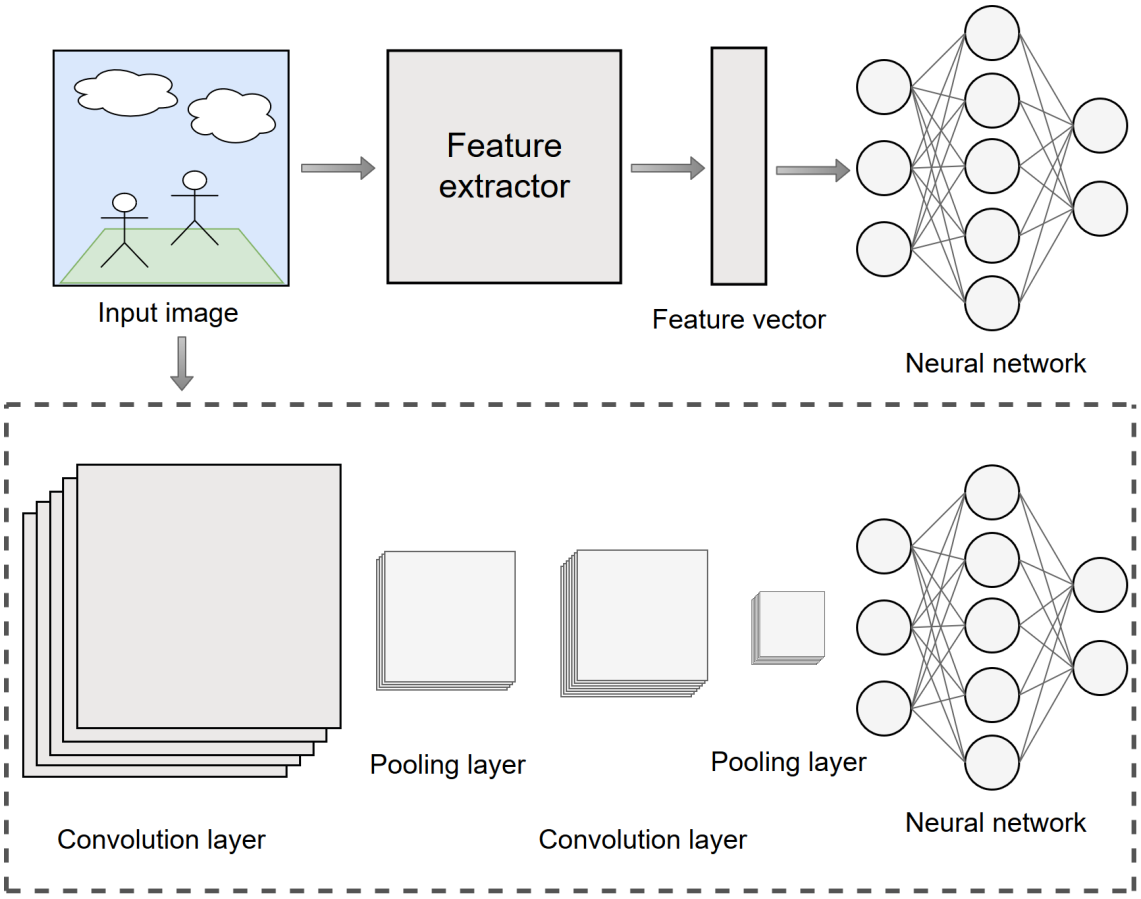


Fig. 1.1: Clasic neurall network vs CNN as classifier.

1.4.1 Convolutional layer

Convolutional layer performs convolution operation on the input of this layer. Output of this layer is called feature map. The convolutional operation is done by tiny neural network with shared weights. For example, the input of convolutional layer is a standard RGB image. It has a width, height and it has also a depth – the three colour channels. Above mentioned tiny NN has also width and height, which is called size of the convolutional filter, and it also has various number of outputs. This layer output will be a different picture with different width, different height and different depth. Convolution layer changes input size in following way (1.7), (1.8), (1.9):

$$W_2 = \frac{H_1 - F_w + 2P}{S} + 1 \tag{1.7}$$

$$H_2 = \frac{H_1 - F_h + 2P}{S} + 1 \tag{1.8}$$

$$D_2 = K \tag{1.9}$$

Where W_2 is the width of convolutional layer output, W_1 is the width of convolutional layer input. F_w represents the width of convolutional kernel – width of convolutional filter. P stands for the size of padding. S is the size of stride. H_2 is the height of convolutional layer output, H_1 is the height of convolutional layer input. F_h represents the height of convolutional kernel. D_2 is the depth of the convolutional layer output and it equals to K – number of small neural network output which was used to create the convolution operation.

Due to a high computational difficulty there is an idea of $1x1$ convolution. It is not convolution at all, but it is used to change depth of the output. $1x1$ convolutions are used to compute depth reductions before the more expensive convolutions. Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose.

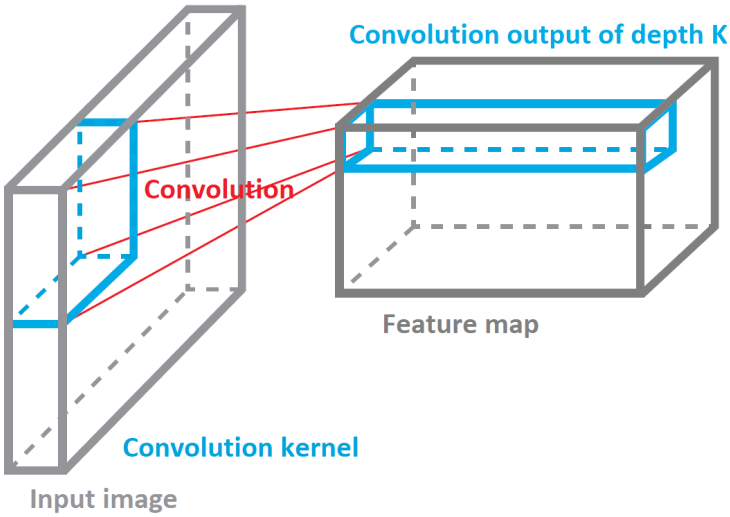


Fig. 1.2: Convolution principle.

1.4.2 Pooling layer

The goal of the pooling layer is to reduce feature map size. The pooling function is applied to all feature maps to reduce their size – it is a form of non-linear down-sampling. There is several of pooling functions, but the most commonly used is the Max Pooling. Each feature map is divided into non-overlapping rectangle regions. For each region, the maximum value is selected, and the remaining are discarded.

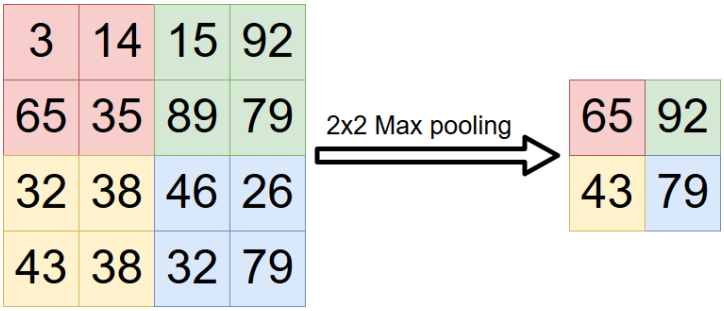


Fig. 1.3: Max Pooling principle.

This layer output size is (1.10), (1.11) and (1.12):

$$W_3 = \frac{H_2 - F_w + 2P}{S} + 1 \tag{1.10}$$

$$H_3 = \frac{H_2 - F_h + 2P}{S} + 1 \tag{1.11}$$

$$D_3 = D_2 \tag{1.12}$$

Where W_3 and H_3 are output width and height of the pooling layer. F_w and F_h are the width and height of the pooling filter. D_2 is the output depth of the pooling layer.

Another option is to use Average pooling or a linear combination of feature map values instead of Max pooling.

1.4.3 Fully connected layer

Fully connected layers are situated at the top of the mentioned pyramid structure. It is used for input classification. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural network.

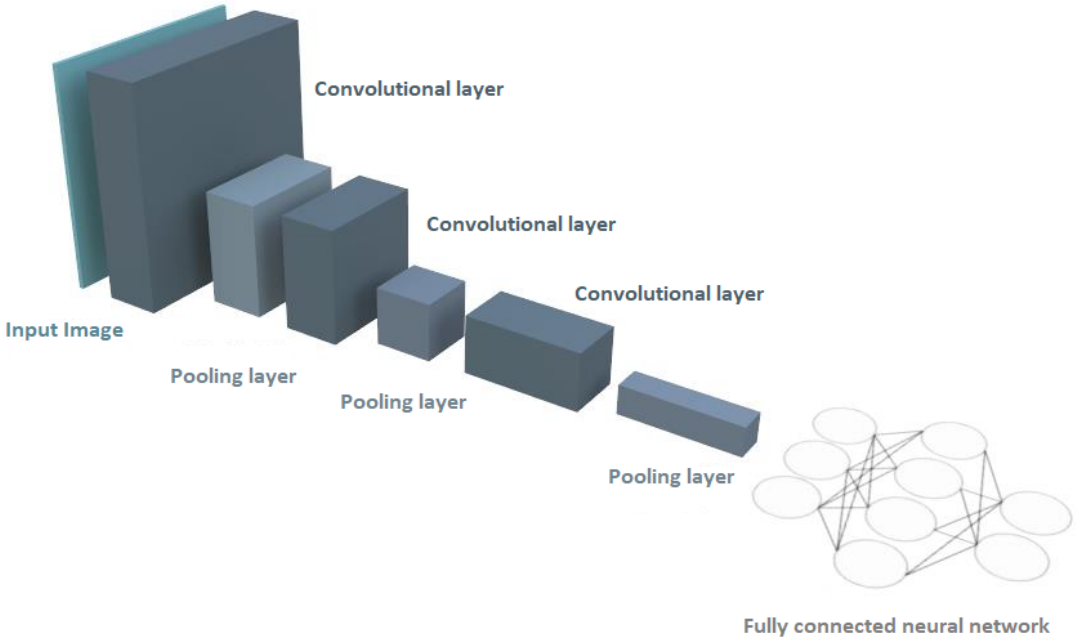


Fig. 1.4: Pyramid layers stacking.

1.5 The Sliding Windows algorithm

So far, we've been discussing the use of CNN as a classifier. Object localization via CNN is very similar to classification. The last layer contains, in addition to the neurons for classification, also the neurons that determine the position of the object. This method is suitable for only one object at a time. To detect multiple objects in the same image it is needed to use the Sliding windows algorithm or the convolutional neural network meta-architecture.

The idea of the Sliding Windows algorithm is very simple. The classic CNN is trained for classification task. For solving the detection problem, it is needed to pick a certain window – part of the image we want to detect in - and run the whole CNN for detection in this window to classify if there is an object or not. In the next step we repeat this process by moving this window across the entire image.

To detect objects of different sizes, we can use several windows with different sizes. This is very simple solution, but it has high computation costs and one object can be detected multiple times depending on the size and stride of the sliding windows. To prevent multiple detection of one object, the Non-max Suppression can be used.

To lower the computation costs the convolutional implementation of the Sliding windows can be used. What this implementation does, is replacing those fully connected layers for classification by convolutional layers. To achieve the same function there is a special layer called Flatten. Flatten layer will literally flatten the input matrix into a vector. This layer is the equivalent of the input layer in the classic fully connected neural network. Next layers of fully connected network are created by $1x1$ convolution with depth of the filter corresponding to number of neurons in the original fully connected layer.

2 STATE-OF-THE-ART

This chapter deals with different architectures of CNN for person detection. We will discuss some of well-known architecture models for classification, which can be used as a backbone convolutional neural network in meta-architectures and later on in this chapter we will discuss meta-architectures for object detection. We will take a closer look at Faster R-CNN, Single Shot Multibox Detector (SSD) and YOLO detector since they are the State-of-the-Art in object detection.

2.1 AlexNet

AlexNet is a well-known large convolutional neural network named after its creator Alex Krizhevzky. It was designed to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into 1,000 different classes and it was published by Alex Krizhevzky, Ilya Sutskever and Krizhevsky's PhD advisor Geoffrey Hinton – all from University of Torronto [11]. This architecture achieved top-1 and top-5 error rates of 37.5 % and 17.0 % which was considerably better than the previous State-of-the-Art.

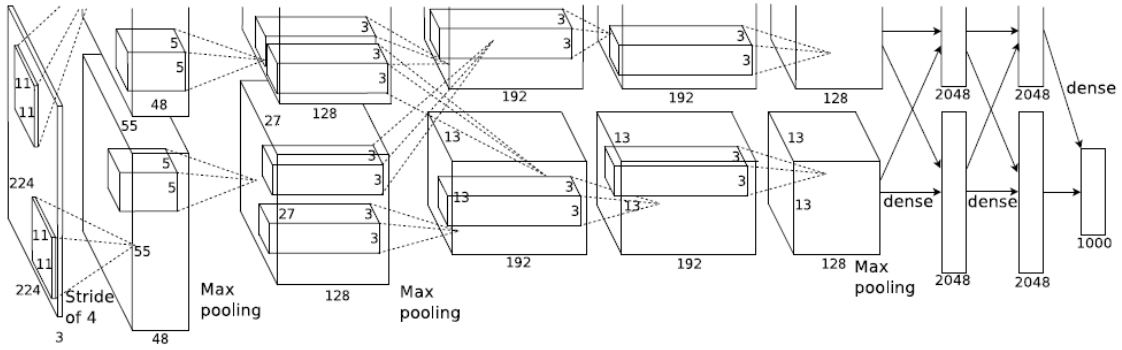


Fig. 2.1: Structure of the original AlexNet CNN [11].

The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of them are followed by max-pooling layers, and three fully-connected layers with a final 1000-way SoftMax. Graphical representation of AlexNet is shown in the figure 2.1. There are two pipelines, because of using two GPUs to train the network. Neurons used in this architecture use non-saturating ReLU function. There are two main features to reduce overfitting in the fully-connected layers. The first one is data augmentation, which is the easiest and most common method. The second one is called Dropout. It consists of setting the output of each hidden neuron to zero with probability 50 %. So, every time

an input is presented, the neural network samples a different architecture, but all these architectures share weights. It is basically the adaptation of different models combining prediction which has way less computational cost.

2.2 VGG nets

VGG nets was presented in Very deep convolutional networks for large-scale image recognition paper [12] by Karen Simonyan and Andrew Zisserman from the Oxford University. This work investigates the effect of convolutional network depth on its accuracy in the large-scale image recognition setting. Their main contribution was an evaluation of networks with increasing depth using an architecture with 3Ö3 convolution filters, which shows that a significant improve comes with the depth of 16–19 layers. The best two architectures were used to contest in ImageNet Challenge 2014 and they achieved the first and the second places in the localisation and classification tracks. The input image is processed by several convolutional layers with stride of 1 pixel. Five convolutional layers is followed by max-pooling layers, which reduce size of feature map by 50 %. At the end the three Fully-Connected (FC) layers can be found. The final layer is the soft-max layer. All hidden layers are equipped with ReLU. The VGG16 architecture tested in this paper is displayed in the figure 2.2.

VGG nets are commonly used for extracting features from images. While the weights are publicly available, VGG nets can serve as the basis for building more advanced architectures.

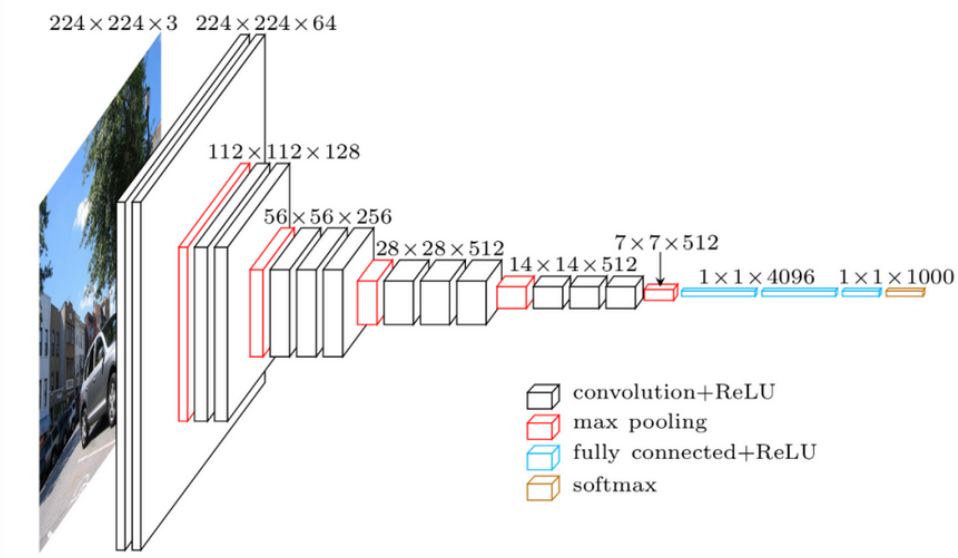


Fig. 2.2: Structure of the VGG16 CNN [13].

2.3 GoogleNet

GoogleNet won ILSVRC 2014 competition. Paper [14] was published by Christian Szegedy et al from Google. GoogleNet is a 22 layers deep convolutional neural network. This architecture brings innovation element called inception module, which is based on several very small convolutions in order to reduce number of parameters.

2.4 Residual Neural Network

Residual Neural Network (ResNet) was introduced at the ILSVRC 2015. Paper [15] was published by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun from Microsoft Research. They introduced innovative architecture with bypass connections and features heavy batch normalization. This network was able to outperform human-level performance at ILSVRC 2015 competition.

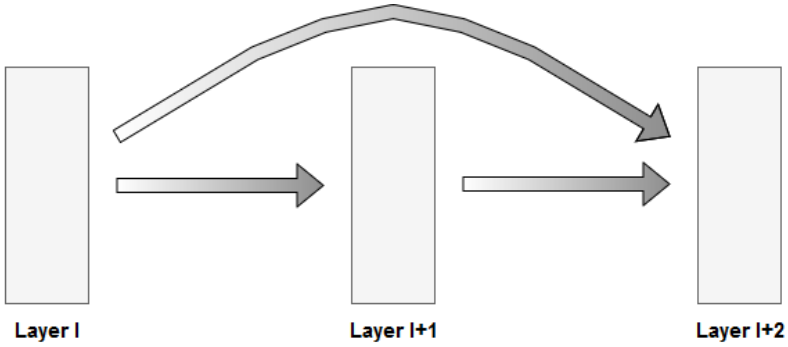


Fig. 2.3: Principle of the layer skipping connection.

2.5 R-CNN family

In the R-CNN family there are meta-architectures of convolutional neural networks which are based on R-CNN. In general, each extension replaces the previous solution. However, R-FCN and Mask R-CNN are already sufficiently different and therefore can be considered as other solutions.

2.5.1 R-CNN

The Region-based Convolutional Neural Networks - R-CNN - was published in Rich feature hierarchies for accurate object detection and semantic segmentation [16]. The R-CNN unlike previously mentioned architectures is used for object detection instead of object classification. It consists of three modules. First one is responsible for generating category-independent region proposals – Region of Interest (RoI). These RoIs are given to CNN (AlexNet in this paper), which output is fixed-size vector of features. Last module is a set of linear SVMs, trained for each class independently.

Despite improved accuracy, this method is not commonly used. The reason for that is its computational costs. However, this method gave rise of a R-CNN family.

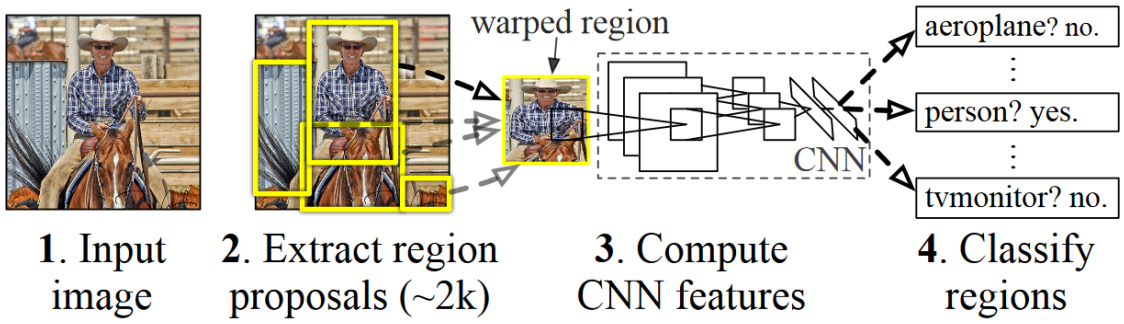


Fig. 2.4: The R-CNN architecture [16].

2.5.2 Fast R-CNN

Fast R-CNN [17] is based on R-CNN. It brings several innovations to improve speed and accuracy. Another undisputed advantage is the end-to-end learning. The graphical representation of a Fast R-CNN is displayed in the picture 2.5.

It has a very similar approach as R-CNN. Main difference between these two algorithms is, that in Fast R-CNN the input image is fed into CNN to generate feature map. From the resulting feature maps the regions of proposals are identified and warped into squares. In the next step these regions are handed to RoI Pooling layer, where each individual region is pooled into fixed-size feature map and then mapped to a feature vector by fully connected layers. The output for each RoI is divided into two vectors. First one is a vector of SoftMax probabilities, the second one contains per-class bounding-box regression offsets.

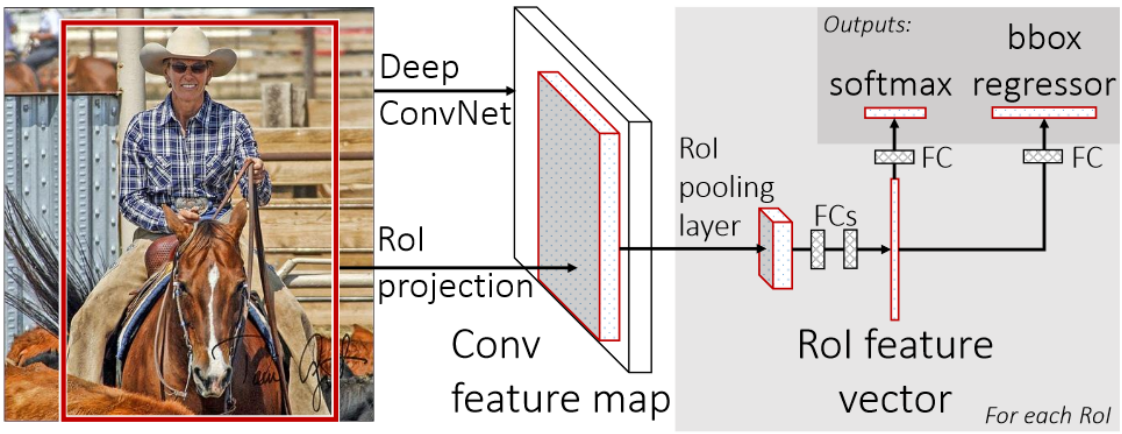


Fig. 2.5: The Fast R-CNN architecture [17].

2.5.3 Faster R-CNN

Faster R-CNN is another improvement of R-CNN. It was published in [18] by Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun. Faster R-CNN brought significant acceleration of detection speed that allows a real time detection and furthermore the precision of object detection achieved accuracy of current State-of-the-Art detectors. These are the reasons, why this architecture is one of the most commonly used architectures nowadays. Faster R-CNN changed the original algorithm for region proposal. The reason for this change is simple. Original region proposal methods are implemented on CPU, while CNNs are implemented for GPUs. Faster R-CNN computes region proposals with a deep CNN. The whole architecture is displayed in the figure 2.6.

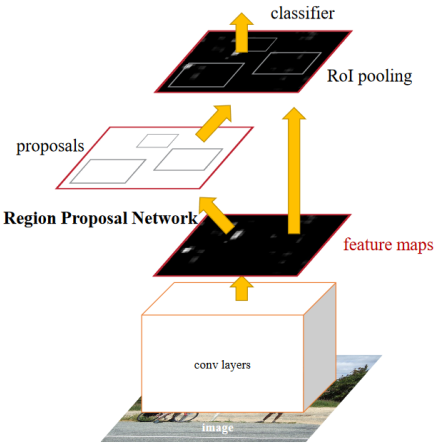


Fig. 2.6: The Fasert R-CNN architecture [18].

Region Proposal Network (RPN) is responsible for generating object proposals with objectness score for each input image. RPN is much faster and also has better accuracy then the original region proposal because it learns itself to classify foreground and background. RPN is located behind the last convolutional layer. It consists of a fully convolutional network (like VGG e.g.). Sliding window algorithm with a small network is used on the feature map generated by the last shared convolutional layer of backbone network. Each sliding window is mapped to a lower-dimensional feature which is passed to two fully-connected layers. The first one is a box-regression layer and the second one is a box-classification layer. Each sliding window generates several anchors – small regions with different $height/width$. Each anchor is centered in the window. During training stage, these anchors are rated by Intersection over Union (IoU). If the IoU of the selected anchor is higher than the threshold for positive, the anchor is marked as positive. Else if the IoU of the selected anchor is lower than the threshold for negative, the anchor is marked as negative. If the IoU of the selected anchor is between these thresholds, the anchor is ignored.

2.5.4 R-FCN

R-FCN was published by Jifeng Dai, Yi Li, Kaiming He and Jian Sun in [19]. R-FCN stands for Region-based Fully Convolutional Network. This architecture is based on Faster R-CNN, but it is fully convolutional. It adopts the two stages strategy for the object detection consisting of the region proposal via RPN and classification of created regions of interest. Given the proposal regions (RoIs), the R-FCN architecture is designed to classify the RoIs into object categories and background. In R-FCN, all learnable weight layers are convolutional and are computed on the entire image. R-FCN ends with a position-sensitive RoI pooling layer which aggregates the outputs of the last convolutional layer and generates scores for each RoI. Position-sensitive RoI layer conducts selective pooling. With the end-to-end training, this RoI layer operates the last convolutional layer to learn specialized position-sensitive score maps. R-FCN uses modified ResNet-101. Basically, it uses only the convolutional layers to compute feature maps. The R-FCN architecture is displayed in the figure (2.7).

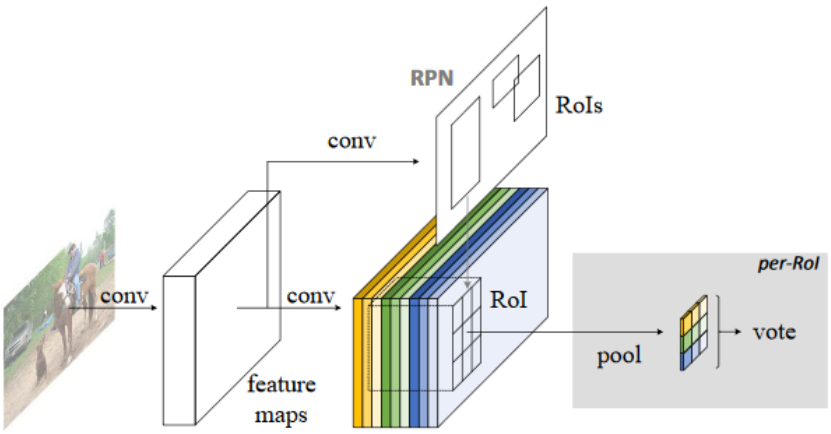


Fig. 2.7: The R-FCN architecture [19].

2.5.5 Mask R-CNN

Mask R-CNN [20] is the extension of Faster R-CNN made by Facebook AI Research team. The extension of Faster R-CNN lies in adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. The authors report the speed of about 5 frames per second.

Mask R-CNN uses the same stages as Faster R-CNN but adds the third output – a binary mask for every single RoI. The mask encodes the object spatial layout and therefore it can be addressed by the pixel-to-pixel correspondence provided by convolutions. As the pixel level segmentation is included, the Mask R-CNN can be much more precise while distinguish which parts of RoI belong to the object.

2.6 SSD: Single Shot Multibox Detector

SSD is another meta-architecture for the object detection via CNN. It affords different approach to the object detection problem than the R-CNN family. As the name suggests, this method uses only one deep neural network to solve detection problem. The advantage of the SSD is its simplicity, which is related to the speed of this method. Accuracy of this meta-architecture is competitive in the comparison to the other State-of-the-Art methods. This determines it for use in real-time and embedded applications.

Originally SSD was published [21] by Wei Liu et al. SSD architecture is displayed in the figure 2.8. SSD originally uses truncated VGG16 CNN as backbone. SSD deforms the input image to fixed size of $300x300$ pixels or $512x512$ pixels because of backbone CNN architecture. There are no fully connected layers in the VGG16

but the last convolutional layer is connected to SSD's convolutional feature layers instead. These layers decrease their sizes and allow the multiple sized objects detection. Each feature layer produces a fixed set of predictions via the set of small convolutional filters. These filters produce either the score for classification or the offset for bounding box since the default bounding boxes are fixed.

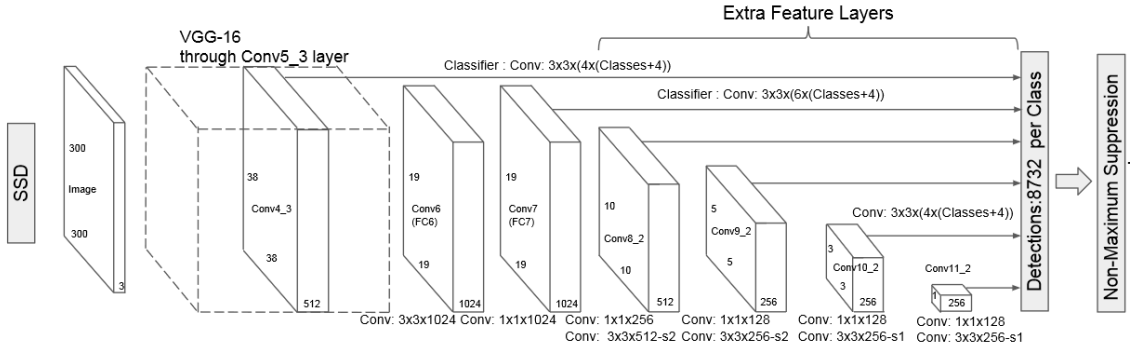


Fig. 2.8: The Single Shot Multibox Detector architecture [21].

As it has been said, SSD has a competitive accuracy, but it is not doing well on small objects. To improve accuracy while detecting small object in the image, several improvements has been included like data augmentation and the à trous algorithm. The SSD creators also mentioned the number of feature layers. But the small object detection problem has not been completely solved yet.

2.7 YOLO: You Only Look Once

YOLO is another single shot detector. YOLO detector originated earlier than SSD, but this method is still being upgraded. Nowadays the third version is actual. YOLO was originally published [22] by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi from the University of Washington, Allen Institute for AI and Facebook AI Research. As other single shot detectors, YOLO uses single deep CNN for both classification and detection.

The advantage of this method is that unlike sliding window and R-CNN family, the YOLO method sees the entire image during training stage so it can see the object in full background context. It divides the input image into an S x S grid. If the centre of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each cell defines B bounding boxes and score for each class. The confidence scores will be zero, if there is no object in the grid cell. Otherwise the confidence scores will be equal the IoU between the predicted box and the ground truth. Each cell produces the class probability either. The whole YOLO architecture

has 24 convolutional layers, followed by two fully connected layers. See the figure 2.9 for more details.

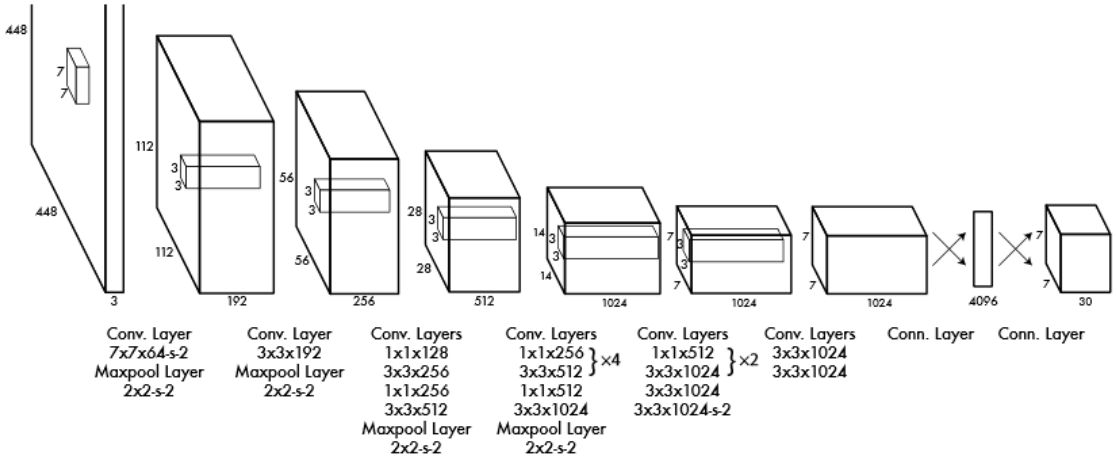


Fig. 2.9: The original YOLO architecture [22].

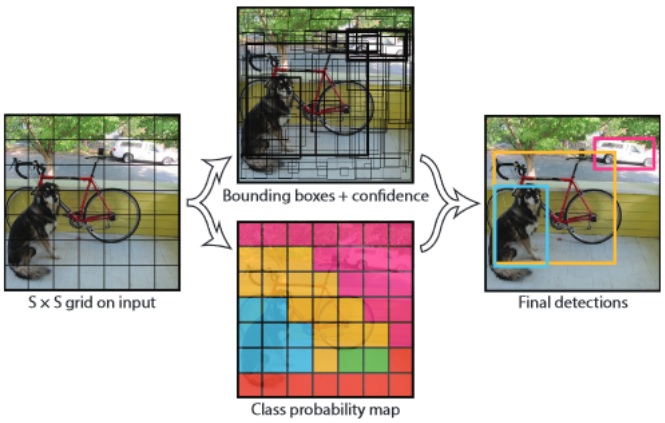


Fig. 2.10: The original YOLO principle [22].

YOLO is not great in detecting multiple small object close to each other (like a flock of birds etc.). YOLO also makes multiple predictions of the same object. Non-maximum suppression can be used to remove them. As it has been said, YOLO architecture is being still improved, because the SSD is its main competitor.

YOLOv2 also known as YOLO 9000 [23] brings significant improvement of speed and accuracy by batch normalization in convolutional layers. It also changes the training stage, bounding box predictions and several other improvements. The latest version of YOLO named YOLOv3 comes with even more upgrades. It was published by Joseph Redmon and Ali Farhadi in [24]. It has the same accuracy as SSD, but it is three times faster. YOLOv3 uses Darknet53 as the backbone CNN, which is

originally a 53 layers deep convolutional neural network. Another 53 layers are added for detection purpose. YOLOv3 also predicts ten-times more boxes than YOLO v2. These are the main reasons, why YOLOv3 is slower than YOLOv2. YOLOv3 makes predictions at three scales. They are done by image downsampling. This makes the small objects easier to be detected.

2.8 RetinaNet

RetinaNet was published by Tsung-Yi Lin et al in [25]. RetinaNet is one stage detector like SSD or YOLO are. It brings novel Focal Loss which focuses training on a sparse set of hard examples and prevents the vast number of easy negatives from overwhelming the detector during training. RetinaNet share lots of similarities with other meta-architectures, like anchors from R-CNN family or feature pyramid from SSD. Figure 2.11 shows the principle of this method. This method aims to fill the gap in accuracy between the single shot detectors and two stage detectors, while achieving better speed.

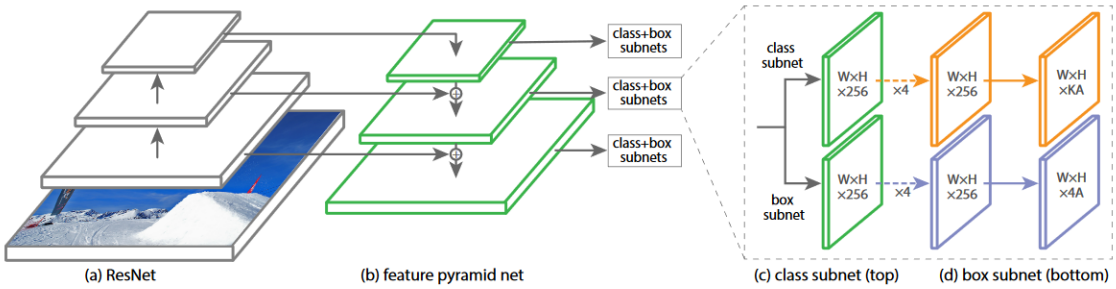


Fig. 2.11: The RetinaNet architecture [25].

RetinaNet uses ResNet CNN architecture as the backbone architecture. On the top of the ResNet there is a Feature Pyramid Network which is used to generate a rich, multi-scale feature pyramid. There are two subnetworks attached by RetinaNet. First of these subnetworks is for classifying anchor boxes, second subnetwork is responsible for the ground-truth object boxes.

2.9 Meta-architectures comparison

There is no easy way to choose which architecture is better because each of the architectures fits into another application. The architecture choice finally depends on the application. This chapter aims to compare State-of-the-Art architectures in terms of speed, accuracy and their use for person detection.

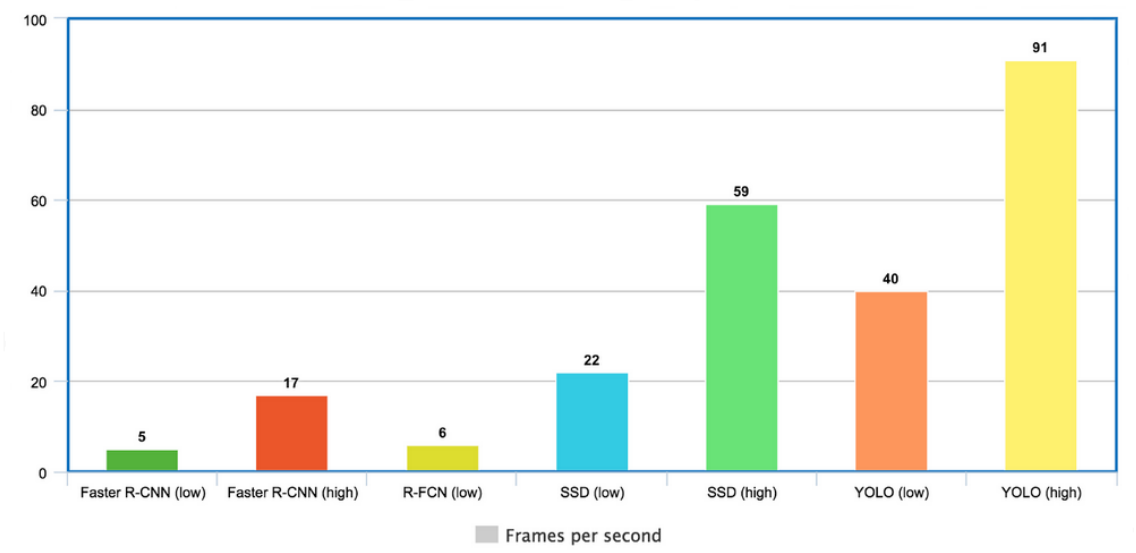


Fig. 2.12: The meta-architecture comparison in term of speed [26].

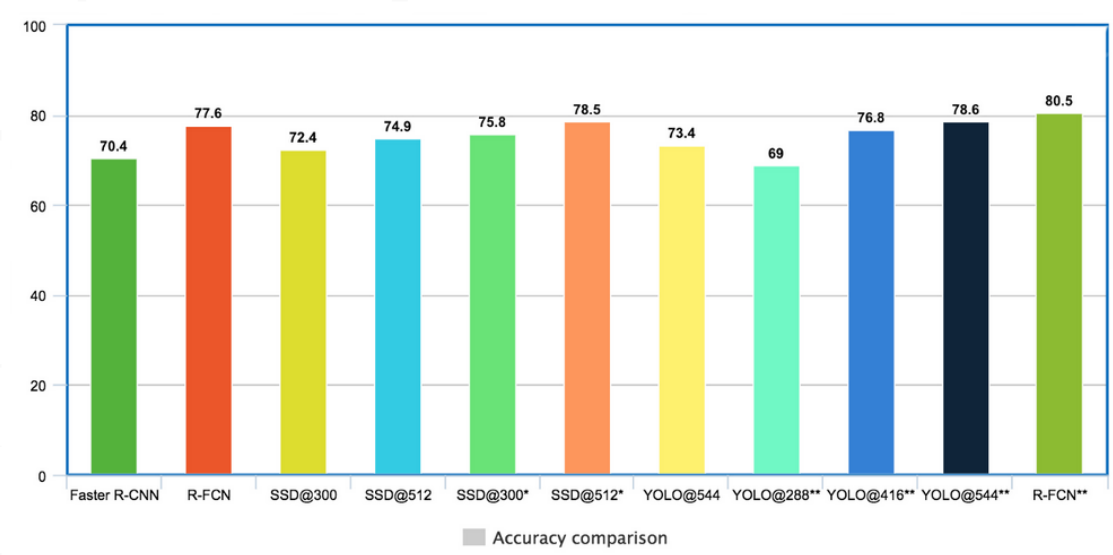


Fig. 2.13: The meta-architecture comparison in term of accuracy on Pascal VOC [26].

In the figure 2.12 speed of meta-architectures is displayed according to [26]. The chart shows the best and the worst fps performance. You can see that YOLO achieved the best performance. The accuracy of each method on the COCO dataset is captured in the figure 2.13. Both speed and accuracy were measured on Pascal VOC 2007 dataset. The second accuracy comparison was measured on COCO dataset.

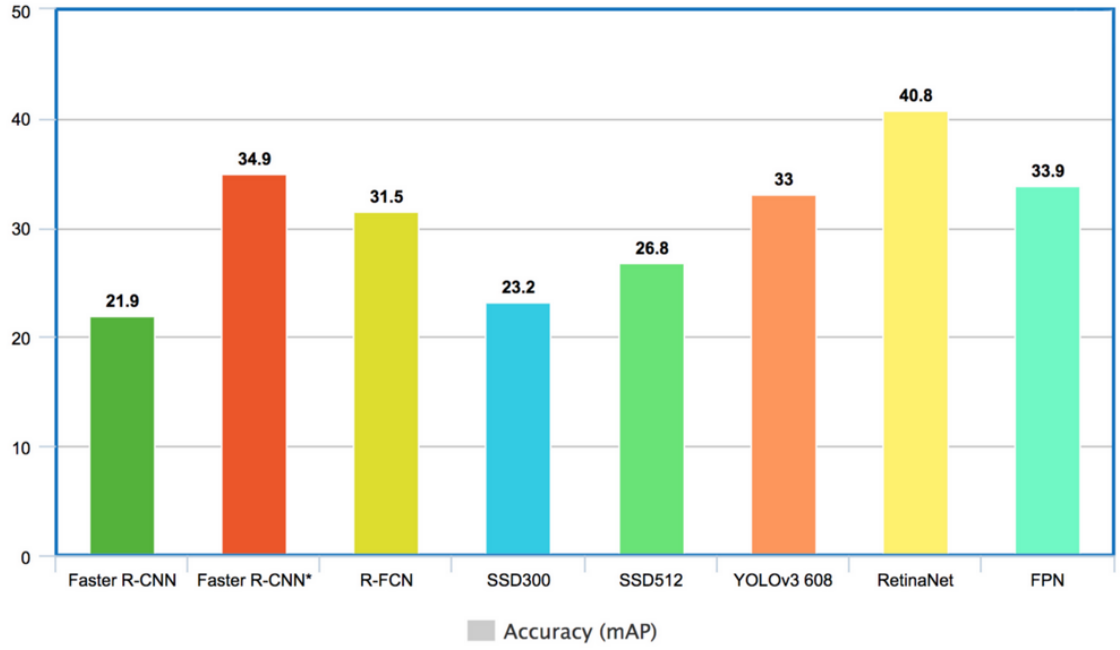


Fig. 2.14: The meta-architecture comparison in term of accuracy on COCO [26].

These results correspond to the results from the paper [6]. The authors of this paper compare how the 13 different models are doing at indoor person detection task. Dataset for this measuring consists of over 10 thousand indoor images captured in shopping malls, retails and stores.

Tab. 2.1: Comparison of 13 methods according to [6].

#	Model	Framework	AP IoU=0.95	IoU=0.50	GPU latency [s]
1	Faster R-CNN (Resnet101)	Tensorflow	0.245	0.476	0.232
2	YOLOv3-416	Darknet	0.143	0.367	0.017
3	Faster R-CNN (InceptionResNet-v2)	Tensorflow	0.317	0.557	0.478
4	YOLOv2-608	Darknet	0.198	0.463	0.035
5	Tiny YOLO-416	Darknet	0.035	0.116	0.011
6	SSD (Mobilenet v1	Tensorflow	0.094	0.233	0.030
7	SSD (VGG 300)	Tensorflow	0.148	0.307	0.015
8	SSD (VGG 500)	Tensorflow	0.183	0.403	0.026
9	R-FCN (ResNet-101)	Tensorflow	0.246	0.486	0.131
10	Tiny YOLO-608	Darknet	0.006	0.185	0.025
11	SSD (Inception ResNet-v2)	Tensorflow	0.116	0.267	0.04
12	SqueezeDet	Tensorflow	0.003	0.012	0.027
13	R-FCN	Tensorflow	0.124	0.319	0.084

Experimental results indicate that Tiny YOLO-416 and SSD (VGG-300) are the fastest and Faster-RCNN (Inception ResNet-v2) and R-FCN (ResNet-101) are the most accurate detectors investigated in this study. Further analysis shows that YOLO v3-416 delivers relatively accurate results in a reasonable amount of time, which makes it an ideal model for person detection in embedded platform. It is generally said that Faster R-CNN is the best option, if accuracy is the main criterium and there is no need for real-time speed. On the other hand, if speed is the main criterium, SSD and YOLO single shots detectors outperforms other solutions. YOLOv3 makes a compromise between accuracy and speed which makes it ideal for pedestrian detection.

2.9.1 Faster R-CNN in pedestrian detection

The authors of this paper [4] investigated the possibilities of Faster R-CNN architecture in pedestrian detection task. They outperformed the hybrid method which uses hand-crafted and deep convolutional features by using modified Faster R-CNN architecture. The modifications lie in using the à trous algorithm and replacing the original classifier by boosted forest classifier. They also investigate that data

augmentation and boostrap method brings a significant improvement in pedestrian detection task. This method achieves 77.12 mAP on KITTI dataset. Detection time was 0.5 s per image on Tesla K40 GPU.

2.9.2 SSD in pedestrian detection

The authors of this study [3] constructed a fully convolutional network for pedestrian detection. Base architecture for this study was the Single Shot Multibox Detector. The low-level convolutional features of created architecture are used to small-scale pedestrian detection and high-level convolutional features aim for large-scale pedestrians. There are more improvements of original SSD like changes in loss function and utilizing prediction boxes with a special log average aspect ratio instead of multiple default boxes. This method can significantly improve recall rate of pedestrians with a different scales and levels of occlusion.

2.9.3 YOLO in pedestrian detection

This paper [27] focuses on pedestrian detection using modified YOLOv2 architecture. In this paper, three Passthrough layers are added to the YOLO v2 network to extract the shallow layer pedestrian features, and the shallow layer features extracted from the Route layer of the original algorithm are improved from the 16th layer to the 12th layer, combine shallow layer features with deep layer features to extract more fine-grained features. These changes improve missrate to 10.05 % while achieving 25 fps.

3 DATASET

Access to a high-quality dataset is one of the most important assumptions of object detection success while using CNN. Since these algorithms are supervised learning type, the quality of dataset used for learning is critical for correct detection. When selecting the appropriate data for the learning process, the application environment for which the model is created is taken into account. The dataset must be sufficiently large for the model to handle generalization. For example, for person detection in a self-driving car application, you need to have a sufficiently diverse scene. Different environments (city, country e.g.), different weather conditions, but also different persons such as color of cloth, relative orientation to camera and other aspects. Since cyclists, motorbike users and pedestrians have different traffic rights, it is also very important to have images where are cyclists and motorbikers in order to teach model, that these are not pedestrians.

3.1 Publicly available datasets for person detection

In this chapter, we will introduce you publicly available datasets, which can be used for person detection from bitmap image. Many of these datasets provides either labeled images and lot of tools for working with dataset like labeling images or evaluating of trained models.

3.1.1 COCO dataset

COCO stands for Common Object in Context, which is also a name of original paper [28]. It is a large-scale dataset for object detection, segmentation and caption. It contains about 328,000 images and more than 200,000 of them are labeled. There is 80 object categories and 91 stuff categories. It also contains 250,000 people with labeled key points. This dataset is available at [29].

Fig. 3.1: An example from COCO dataset.

3.1.2 KITTI dataset

Large scale dataset captured by car equipped with two high-resolution color and gray-scale video cameras, Velodyne laser scanner and GPS. Filmed environment contains either urban areas and rural areas including highways. There is up to 15 cars and 30 pedestrians captured in single image. The dataset is divided in three parts. The 2D Object Detection dataset contains 7,481 training images and 7,518 testing images. Second part is a dataset for 3D Object Detection which consists of 7481 training point clouds (and images) and 7,518 testing point clouds (and images). The last part is a Bird's Eye View dataset which consists of 7,481 training point clouds (and images) and 7,518 testing point clouds (and images). In addition, the entire dataset can be divided according to the difficulty as easy, moderate and hard. This division is done by bounding box height, occlusion level and truncation. For pedestrian detection, there is more than 2,000 pedestrians annotated according to the paper [30]. The dataset is publicly available at [31].

Fig. 3.2: An example from KITTI dataset.

3.1.3 Caltech Pedestrian Detection Benchmark

This very large pedestrian dataset consists of 30 Hz video sequences. Video resolution is $640x480$ pixels. The length of video sequences in total is about 10 hours. Video was recorded in urban streets as a view from driving car. There are about 2,300 unique pedestrians captured. The creators of Caltech dataset also provide labelling and evaluating tools. The Caltech Pedestrian Detection Benchmark is available [32].

Fig. 3.3: An example from CALTECH dataset.

3.1.4 Inria Person Dataset

Inria Person Dataset is a dataset collected as a part of research work on detection of upright people in images and video. The dataset is divided in two formats: original images with corresponding annotation files, and positive images in normalized $64x128$ pixel format with original negative images. Images in this dataset were collected from several different sources – form Graz 01 dataset, personal images cropped to highlight person and few images from Google search. [33]. There are 1,805 images for training. Dataset is usually used for training single person detectors.

Fig. 3.4: An example from INRIA dataset.

3.1.5 Penn-Fudan Database for Pedestrian Detection and Segmentation

The whole dataset consists of 170 images with 345 labeled pedestrians, while each image contains at least one pedestrian in it. All pedestrians are walking, so they are straightened up. Images are cropped to size of $180x390$ pixels. This dataset is available at [34].

3.1.6 Pascal VOC dataset

Pascal Visual Object Classes datasets was published for Pascal VOC challenge. The datasets were designed for challenges held between 2005-2012 and are still available. The creators offer both datasets with corresponding annotations and evaluating server. The individual datasets vary both in the range and in the number of classes (since 2007 is the standard of 20 classes). Pascal VOC datasets are available at [35].

Fig. 3.5: An example from Pascal VOC2012 dataset.

3.1.7 CityShapes dataset

Another large dataset from urban environment. There are 30 classes from urban scenes annotated.The dataset was taken as a record from an on-board camera. The pictures were taken in fifty different locations throughout the daytime and in different weather conditions. Volume of this dataset consist of 5,000 annotated images with fine annotations and 20,000 annotated images with coarse annotations. In addition to images and relevant annotations, the creators also provide meta-data such as ambient temperature and GPS coordinates etc. Persons in this dataset are labeled as a person or as a rider.

On the top of this dataset there is third-party annotations set named CityPersons, which consist of 3,475 annotated images. Both CityShapes and CityPersons are available at [36].

3.1.8 BIWI Walking Pedestrians dataset

This dataset specializes on walking pedestrians in a busy scenario. Pedestrians are recorded from bird eye view. Annotations are done manually. Example of this dataset is displayed in the figure 3.7. Both BIWI Walking Pedestrians dataset and Central Pedestrian Crossing Sequences are publicly available at [37].

Because of the bird eye view, this dataset is not suitable for our task.

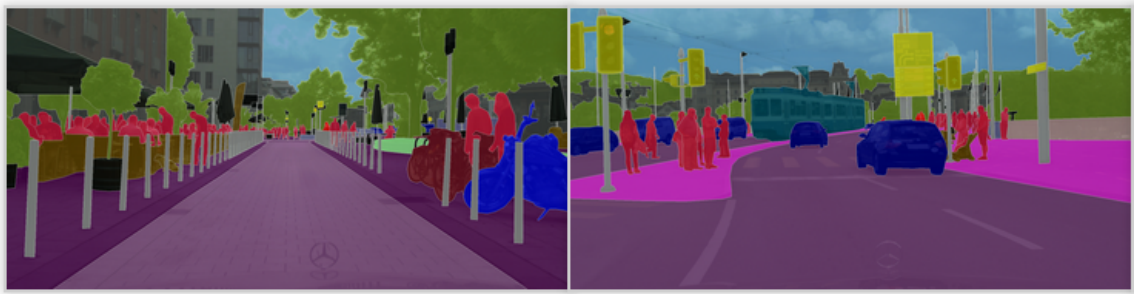


Fig. 3.6: An example from CityShapes dataset.

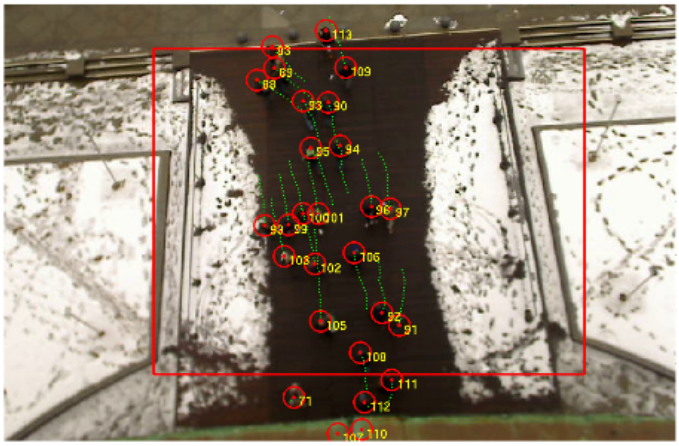


Fig. 3.7: An example from BIWI dataset.

3.1.9 "Central" Pedestrian Crossing Sequences

Three pedestrian crossing sequences used in our ICCV'07 paper. Each sequence comes with ground-truth bounding box annotations for the objects to be tracked, as well as a camera calibration data. The annotation files for the pedestrian crossing sequences contain bounding box annotations for every fourth frame. This dataset is available at [37].

3.1.10 ETHZ Multi-Person Tracking dataset

This dataset was recorded by two cameras mounted on a car. The resolution of taken images is $640x480$ pixels. The dataset is divided into several sequences with more than 5,000 frames in total. The dataset is publicly available at [38].

3.1.11 Mall Dataset

Mall Dataset was collected from publicly accessible web-cam. There is more than 60,000 labeled pedestrians. Each of them is labeled by marking head position. The resolution of taken images is $640x480$ pixels. There are 2,000 frames in total. The Mall Dataset is publicly available at [39].

Fig. 3.8: An example from Mall dataset.

3.1.12 NightOwls dataset

The NightOwls dataset as its name says focuses on nigh traffic environment. According to the paper [40], there are 3 classes annotated - the pedestrians, the bicycle drivers, the motorbike drivers and areas to ignore. Dataset was recorded at the resolution $1024x640$. The dataset volume consists of the 279,000 frames taken during all 4 seasons in all weather conditions at both the dawn and night time. This dataset is not yet publicly available.

3.2 Model error estimation

We can never exactly measure the model accuracy since we cannot test our model on every single possible scene. There are several ways to estimate the model error. We can use estimation just by learning error. Or we can use the HoldOut method, which is very simple. But there are even other methods such as Boostrap and Cross Validation.

3.2.1 Training set data

Training set is used to train the model to recognize relationship between input image and detected object. This part of the dataset is larger than the other parts. It is not precisely determined how much of the entire dataset is to be used as a training

set, but it is usually between 50 % and 75 % of the entire dataset. If we use only the training set, a re-substitution error occurs which leads to real error underestimation.

3.2.2 HoldOut: Testing set data

To avoid the re-substitution error, we can split the entire dataset to training and testing data or to training, testing and validation sets. This method is called Hold-Out. So, the testing set is another part of the dataset, which is not used for learning. The testing dataset is created with equal distribution of different classes of data. It is used to determine how accurate the created model is. The size of this set usually takes from 15 % to 25 % of the entire dataset size.

3.2.3 HoldOut: Validation set

In addition to the training set and testing set, there is optionally the validation dataset in the HoldOut method. This set provides a final estimation of the model error after training and testing stage. This set is for estimating error only, it should not be used for tuning model performance. The size of this set usually takes from 10 % to 25 % of the entire dataset size. The HoldOut method is very simple way to estimate error of machine learning model, but it can be used only if we have large enough dataset.

3.2.4 Boostrap method

Boostrap method is another way to estimate error of the machine learning model when making a prediction on data not included in the training set. It can be used in an advantage if we do not have enough training data, but it can be also used to create robust model in meta-learning.

The whole dataset with N elements is used to generate B training set with N elements by sampling the original dataset with replacement. Error estimation can then be determined according to the equation (3.1).

$$Err_{boot} = \frac{1}{N}\frac{1}{|B|}\sum_{j=1}^{|B|}\sum_{i=1}^{N} LF(y_i, f^{B_j}(x_i)) \tag{3.1}$$

There is more accurate way to use boostraping to error estimation. It is named .632 Boostrap. The difference between classic boostrap and .632 boostrap is, that elements that have not been selected for training are used for testing. There is

approximately 0.368 chance, that a certain element will be not selected for training. The error estimation of this submodel can be then determined as (3.2).

$$Err_{Btest} = \frac{1}{|B|}\sum_{j=1}^{|B|}\frac{1}{|C_j|}\sum_{i=1}^{|C_j|} LF(y_i, f^{B_j}(x_{C_{ji}})) \tag{3.2}$$

The overall error is calculated as (3.3).

$$Err_{.632} = 0.632 Err_{Btest} + 0.368 Err_{boot} \tag{3.3}$$

3.2.5 Cross Validation method

Cross-validation is the method of determining how much the statistical analysis model will influence independent data samples. This method is useful for prediction of unknown samples after prior classification of known samples. Like the Boostrap method, the Cross Validation can be used for either error estimation either for robust model creating in meta-learning.

The principle lies in dividing the entire available dataset into K different sets. The model is then K times trained, while K-1 sets are used for training and the last set is used for testing. The total error of the model is given by the average of the errors of these K models (3.4).

$$Err_{CV} = \frac{1}{K}\sum_{i=1}^{K} Err(y_{K(i)}, f^{K_i}(x_{K_i})) \tag{3.4}$$

3.3 Dataset pre-processing

As it has been said, the quality of the resulting model depends on the quality of the dataset. Data pre-processing can make a huge difference in the learning process and sometimes it is necessary since the CNNs require the fixed size input. Pre-processing includes for example an image scaling or cropping, normalization, data augmentation, uniting the annotations format and so on. Every operation on dataset before the training stage can be called pre-processing.

3.3.1 Normalization

Normalization serves to describe the data in a suitable format. In the simplest case, it is mapping any interval into the interval $< 0, 1 >$ (3.5). However, this can be a more complex process.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.5}$$

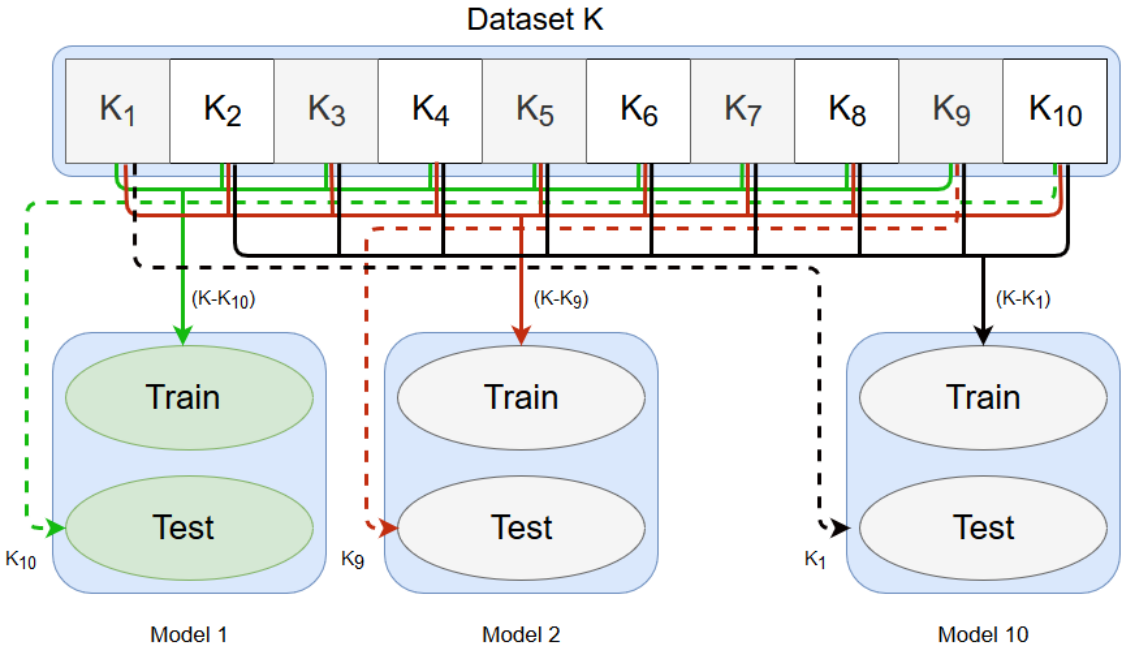


Fig. 3.9: The Cross Validation principle.

3.3.2 Data augmentation

The data augmentation is a pre-processing technique involving augmentation of existing datasets by information derived from the same dataset. It is useful to reduce over-fitting and it also generates more data by which the training set is expanded. To achieve this geometric transformations, blurring and averaging are typically used.

3.3.3 Negative and Hard negative mining

The negative mining is used for teaching CNN what is the object (positive instance) and what is the background (negative instance). It can be done by generating the bounding boxes on background and labeling them as negative (no object). The hard negative mining is used to prevent the false positive detection. It is basically done via marking bounding boxes over background which was detected and ranked as false positive and then retrain the model.

4 EXPERIMENT DESIGN

This chapter deals with finding the best practice of creating a training dataset. The resulting accuracy of the trained model depends on many factors like type of the meta-architecture, architecture parameters, dataset size and diversity. Finding optimal model is nearly impossible, because we have to use brute-force for going through many-dimensional space. Instead, we will go through this space by gradually changing only one parameter at one time. This iterative method will result in finding a pseudo-optimum model, because it will probably fail to find a global extreme. The pseudo-optimum thus found may also not be valid for different meta-architectures. This is the reason why we decided to test two meta-architectures. We will start training both meta-architectures and after first evaluation we will continue in training only with the meta-architecture that achieves better results. In the end we will compare both architectures trained on dataset with the best performance of the previous results. All the models thus obtained are then subjected to more detailed analysis using different criteria.

4.1 Suitable dataset creation

The basis for a successful model is the data used in training stage. There is no quality model without quality training dataset. We decided to use mainly KITTI dataset with CityPersons dataset because this thesis deals with pedestrian detection from an autonomous vehicle perspective. Both of these datasets were recorded from car, so they perfectly correspond to the given task. In order to make the model more general, Pascal VOC dataset was selected to complement the resulting dataset with images of people in environment other than traffic.

However, these datasets do not contain images taken at night and in bad weather conditions. This is because ordinary cameras are not the most suitable tool for these conditions, and today's self-driving cars do not rely solely on the camera but use other sensors such as Lidar. Assuming we only want to rely on the camera for night-time driving, we need to have the model trained for these conditions as well. For solving this task, we created Small Night Pedestrian Dataset, which we will introduce in next subsection.

4.1.1 Small Night Pedestrian Dataset

As been said, Small Night Pedestrian Dataset was created in order to train model for night scenario. Dataset consist of 227 images with 815 pedestrians annotated in total. The picture 4.1 shows a more detailed view of the distribution of the

pedestrians per image. There are 44 small (max 32 x 32 px), 414 medium (max 96 x 96 px) and 357 large (> 96 x 96 px) pedestrians annotated. Average width of annotated pedestrian is 35.99 px while average height is 101.59 px.

Fig. 4.1: Histogram of pedestrian distribution in images for Small Night Pedestrian Dataset.

Dataset was recorded on several commonly used on board cameras in order to achieve more variety in video quality. We also recorded in different times (like early morning, sunset, late night) and different weather conditions including heavy raining. Dataset covers mainly small-town areas but you can find images from city center and countryside environment in there.

Fig. 4.2: Image from the Small Night Pedestrian Dataset.

We hand annotated all dataset images with LabelIMG tool [41]. Similar to KITTI dataset we also annotated the whole pedestrian and not only the visible part if any part of the pedestrian was covered by another object. We also decided to not annotate very small pedestrians in background with either width or height less than 5 px. The smallest pedestrian width is 5 px and the smalles pedestrian height is 17 px.

Every image was cropped in order to reduce its size and remove visible part of the car body from which the images were taken. Fixed cropping dimensions are not retained for the best variety.

4.1.2 Data splitting

We decided to use HoldOut with eval set method. Because training stage is computationally challenging, using Boostrap or Cross-Validation method will be to much time consuming.

Training set consist of 5,000 images collected of KITTI, CityPersons, Small Night Pedestrian Dataset and Pascal VOC datasets. Test dataset consists of all selected except Pascal VOC. But there are mostly images from KITTI dataset.

The eval dataset is divided into three parts. We decided to test trained models on each dataset separately. So, first part consists of KITTI images, second part consists of CityPersons images and the last part consists of Small Night Pedestrian Dataset images. These parts do not contain the same number of images. This division was made in order to better understand models behavior.

4.1.3 Dataset analyzing

The whole experiment was done with several training dataset sizes. Datasets have a size of 10, 100, 500, 800, 2,000 and 5,000 images. This subchapter aims to analyse the characteristics of each dataset.

The 10-frame dataset composes of randomly selected images from all datasets used. There are 28 pedestrians. The average pedestrian size is 43.71 x 117.93 px. There is 1 small (max 32 x 32 px), 9 medium (max 96 x 96 px) and 18 large ($>$ 96 x 96 px) pedestrians.

Fig. 4.3: Histogram of pedestrian distribution in images for 10-frames dataset.

The 100-frame dataset was compiled by randomly selecting images from all datasets used. There are 351 pedestrians. The average size of annotated pedestrian is 46.44 x 109.63 px. There are 29 small, 141 medium and 181 large pedestrians.

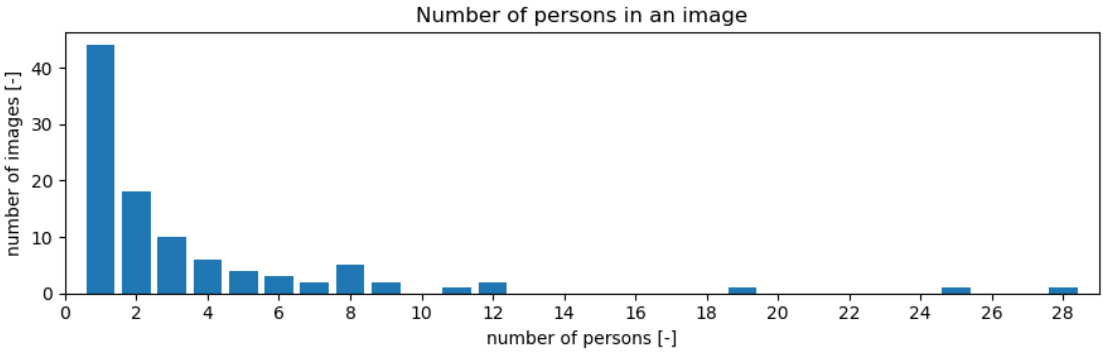


Fig. 4.4: Histogram of pedestrian distribution in images for 100-frames dataset.

The 500-image dataset consists of only a KITTI dataset. There are 1,238 pedestrians. The average pedestrian size is 44.08 x 104.70 px. There are 47 small, 606 medium and 585 large pedestrians.

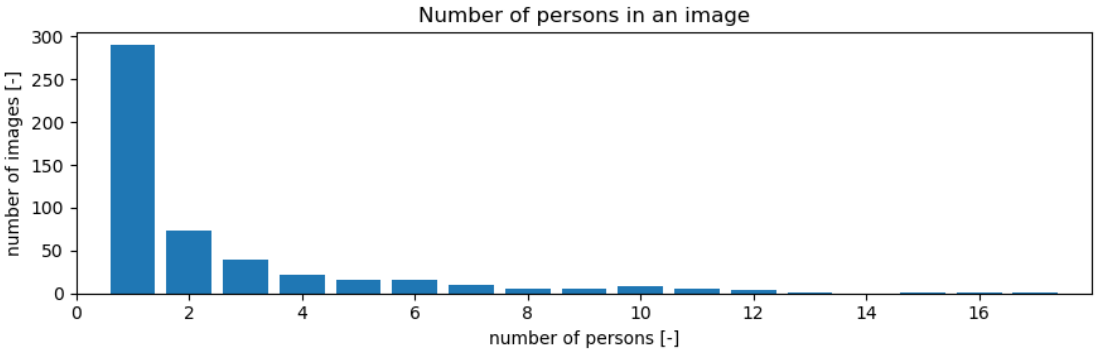


Fig. 4.5: Histogram of pedestrian distribution in images for 500-frames dataset.

The 800-frame dataset consist of images from all datasets used. There are total 3,075 pedestrians annotated. The average pedestrian size is 45.25 x 108.85 px. There are 185 small, 1,496 medium and 1,394 large pedestrians.

One of the larger training datasets is the 2,000-frame dataset. It was compiled by randomly selecting images from all datasets used. There are 7,324 pedestrians. The average pedestrian size is 50.54 x 115.25 px. There are 454 small, 3,531 medium and 3,339 large pedestrians.

The largest training dataset consist of all 5,000 images. All training images from KITTI, CityPersons and Small Night Pedestrian Dataset was used. These images were supplemented with images from Pascal VOC dataset. There are 23,156

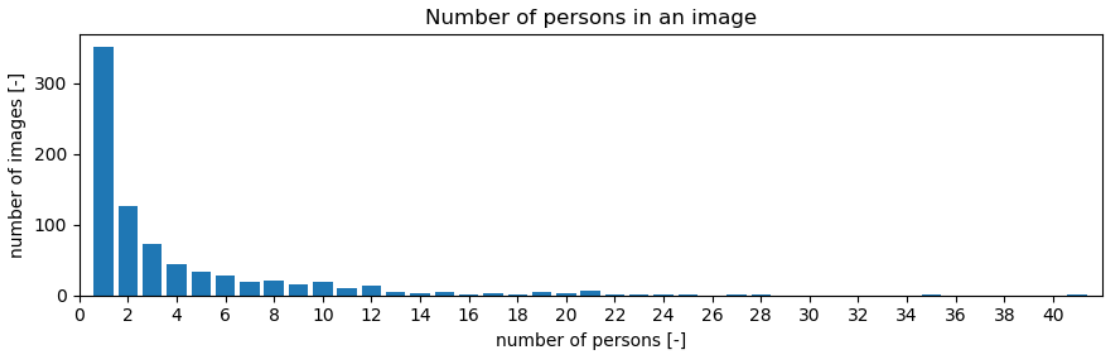


Fig. 4.6: Histogram of pedestrian distribution in images for 800-frames dataset.

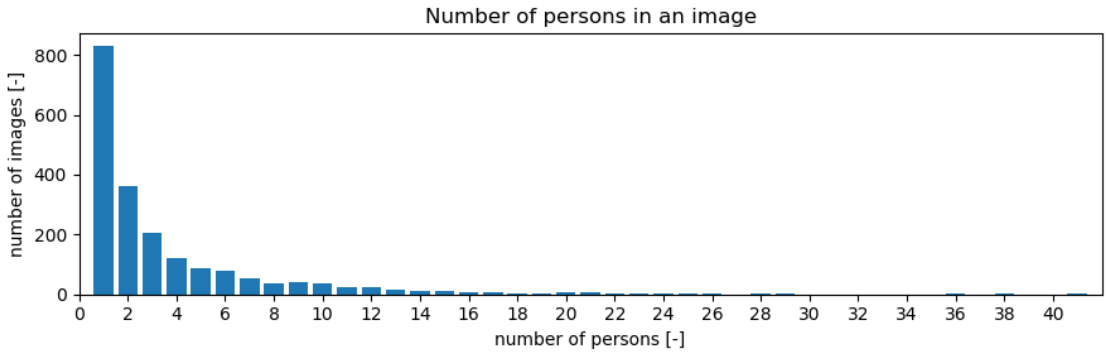


Fig. 4.7: Histogram of pedestrian distribution in images for 2,000-frames dataset.

pedestrians. The average pedestrian size is 50.84 x 116.49 px. There are 1,732 small, 10,949 medium and 10,475 large pedestrians. The histogram of pedestrian distribution in images of this dataset can be found in the appendix.

	A	B	C	D	E	F	G	H
1	filename	width	height	class	xmin	ymin	xmax	ymax
2	frankfurt_000000_000294_leftImg8bit.png	2048	1024	person	947	406	963	445
3	frankfurt_000000_000294_leftImg8bit.png	2048	1024	person	1157	375	1197	473
4	frankfurt_000000_000294_leftImg8bit.png	2048	1024	person	1195	381	1229	464
5	frankfurt_000000_000294_leftImg8bit.png	2048	1024	person	1223	379	1260	469
6	frankfurt_000000_001016_leftImg8bit.png	2048	1024	person	719	325	819	569
7	frankfurt_000000_001016_leftImg8bit.png	2048	1024	person	923	364	984	514
8	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	1055	387	1077	441
9	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	1187	381	1208	432
10	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	1203	384	1224	437
11	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	1227	382	1247	433
12	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	1218	379	1241	437
13	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	1273	372	1308	459
14	frankfurt_000000_001236_leftImg8bit.png	2048	1024	person	991	364	1025	449

Fig. 4.8: An example of csv dataset labels.

The test dataset consists of images from KITTI, CityPersons and Small Night

Pedestrian Dataset. There are 893 pedestrians in 204 images. The average pedestrian size is 44.95 x 111.26 px. There are 68 small, 420 medium and 405 large pedestrians.

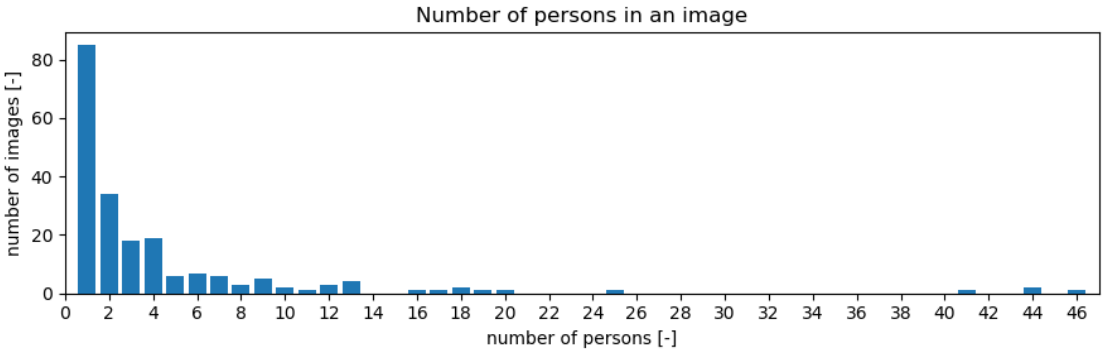


Fig. 4.9: Histogram of pedestrian distribution in images for test dataset.

The eval KITTI dataset consists of 1,179 images with amount of 2,992 pedestrians. The average pedestrian size is 43.66 x 103.21 px. There are 142 small, 1,495 medium and 1,355 large pedestrians.

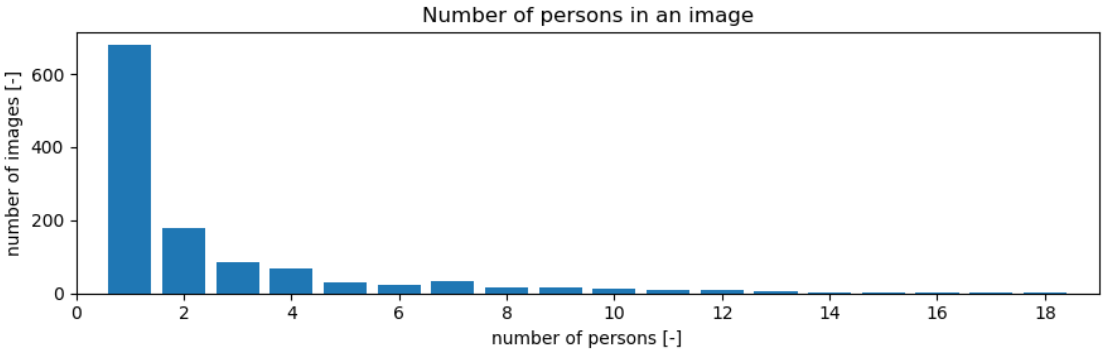


Fig. 4.10: Histogram of pedestrian distribution in images for eval KITTI dataset.

The second evaluation dataset dataset consists of 398 CityPersons images. There are 3,157 pedestrians. The average pedestrian size is 47.47 x 117.23 px. There are 210 small, 1,457 medium and 1,490 large pedestrians.

The last evaluation dataset consists of 54 images. These images belong to Small Night Pedestrian Dataset. There are 163 pedestrians. The average pedestrian size is 49.60 x 143.72 px. There are 4 small, 50 medium and 109 large pedestrians.

Fig. 4.11: Histogram of pedestrian distribution in images for eval CityPersons dataset.

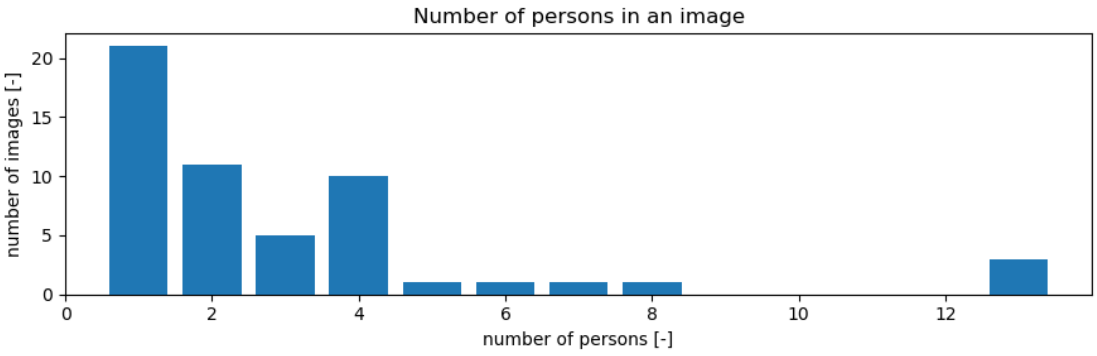


Fig. 4.12: Histogram of pedestrian distribution in images for eval Small Night Pedestrian Dataset.

4.2 Used meta-architectures

Based on the research in the State-of-the-Art chapter we decided to test the Faster R-CNN and the Single Shot Multibox Detector meta architectures. These meta-architectures achieve the best accuracy or detection speed. While machine learning model is used in autonomous cars, we need the best accuracy possible, because one false negative prediction can result in a tragic accident. Cars move at high speed, so detection speed is crucial as well. It is not easy to find a compromise between accuracy and detection speed.

4.2.1 Framework selection and used tools

We chose Tensorflow [42] as a base framework for meta-architectures testing. Tensorflow is the end-to-end open-source platform for machine learning actively developed by Google. It offers multiple level of abstraction from the high-level Keras API for

easy model deploying to the low-level API for more flexibility. Tensor-flow is targeted either for CPUs or GPUs. It also offers several programming language APIs. We had chosen the Python API because Python is the high-level OOP programming language which is suitable for data analyzing. There is also Google's open-source machine learning library the Tensorflow Object Detection API [43], which we decided to use.

Fig. 4.13: NVIDIA Titan X [44].

4.2.2 Implementation

We decided to use the Tensorflow GPU version since most of the machine learning models are optimized for GPUs. For GPU support Tensorflow needs GPU with CUDA Toolkit 9.0 a cuDNN v7.0 installed. GPU must also meet the Compute Capability 3.7 requirement, which makes GPU version of Tensorflow suitable only for NVIDA GPUs.

The whole experiment was done on two computers. One ordinary laptop with 4x Intel i7 (1.8 GHz, turbo boost 4 GHz), 8 Gb RAM and NVIDIA GeForce MX 150 2 GB and one personal computer with 4x Intel i7 (5.6 GHz), 32 Gb RAM and NVIDIA Titan X 12 Gb. Both computers run Windows 10 and Python 3.6.7. Tested version of Tensorflow is tensorflow-gpu 1.12.0. You can see attachment B for short comparison of selected NVIDIA GPUs suitable for Object Detection API with GPU support.

The Tensorflow installation was performed using the pip tool with the command:

```
pip install tensorflow-gpu
```

Then it was necessary to install the necessary dependencies for the Tensorflow Object Detection API:

```
pip install Cython
pip install contextlib2
pip install pillow
pip install lxml
pip install jupyter
pip install matplotlib
```

After installing the necessary dependencies, you must download the Tensorflow Object Detection API. This can be done, for example, by the Git tool or via direct download from the official GitHub page. The next step was to install the COCO API. Because both computers use the Windows 10 operating system it was necessary to use a modified version of the COCO API which can be found on GitHub [45]. Subsequently, it was necessary to compile Protobuf:

```
protoc object_detection/protos/*.proto --python_out=.
```

Note that Protobuf version 3.4.0 is necessary on Windows 10. The last step was creating system path named PYTHONPATH:

```
<Object Detection API>\models\research
<Object Detection API>\models\research\object_detection
<Object Detection API>\models\research\slim
```

```
<Object Detection API>\models\research\slim\datasets
<Object Detection API>\models\research\slim\deployment
<Object Detection API>\models\research\slim\nets
<Object Detection API>\models\research\slim\preprocessing
<Object Detection API>\models\research\slim\scripts
```

Once installation was done, we had to chose pretrained model. We decided to use the Faster R-CNN with ResNet 101 as the feature extractor pretrained on pedestrian and car objects from the KITTI dataset. This model was selected because it was already pre-trained on images from the traffic environment, so the transfer learning time would be much smaller. For SSD there was no traffic environment pretrained model, so we decided to use the SSDLite with the MobileNet_v2 as the feature extractor pretrained on the COCO dataset.

On the Windows 10 operating system we faced several errors which occurred because the Object Detection API library is targeted primary for Linux operating systems. We had to change some of the source files to get everything working. These files are:

```
detection_inference.py
object_detection_evaluation.py
tf_example_parser.py
```

All of these files can be found in the attachment CD in the tensorflow/Code Corections directory.

4.2.3 Created tools

In order to use images from multiple dataset, we had to choose a uniform dataset format and transform images that don't match this format. We decided to use Pascal VOC format of annotations because unlike KITTI annotations, it is easy to read for humans. Another advantage of this format is that each image has the same named xml file, which contains information about objects, but also the entire dataset. This format makes it easy to work with the dataset as a whole.

We managed to create Python library containing useful dataset functions and functions for the subsequent conversion of the dataset to the Tensorflow Object Detection API. In addition to the dataset features, we created scripts to detect pedestrians from the image, live webcam stream and from already recorded video. These scripts can serve as object detectors but also as object detection benchmark. All of these files can be found in attachments. You can see an example from this library - the *xml_to_csv*() function.

```
def xml_to_csv2(path, listpath='suffleList.txt'):
    xml_list = []
    listFiles = []
    with open(listpath) as my_file:
        listFiles = my_file.readlines()

    for xml_file in listFiles:
        xmlPath = ''.join(xml_file)
        xmlPath = xmlPath.strip('\n')
        tree = ET.parse(path+xmlPath+".xml")
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size').find('width').text
                        ),
                     int(root.find('size').find('height').
                        text),
                     member.find('name').text,
                     int(float(member.find('bndbox').find('
                        xmin').text)),
                     int(float(member.find('bndbox').find('
                        ymin').text)),
                     int(float(member.find('bndbox').find('
                        xmax').text)),
                     int(float(member.find('bndbox').find('
                        ymax').text))
                     )
            xml_list.append(value)
    column_name =
    ['filename', 'width', 'height', 'class', 'xmin', 'ymin',
       'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)

    return xml_df

def xml_to_csv(path = "", listpath="suffleList.txt"):
    image_path = os.path.join(os.getcwd(), path)
    xml_df = xml_to_csv2(image_path, listpath)
    xml_df.to_csv('labels.csv', index=None)
    print('Successfully converted xml to csv.')
```

The *xml_to_csv* function takes 2 parameters. The path to the folder where are

the xml annotation files stored and listpath - path to text file containing names of xml annotations. You can create this text file via $makeSuffleFilesTxt$ or $createFileList$ functions from our library. This file determines the order of handing the images to the training algorithm. In order of batch training, which is depends on the order of images, use the $makeSuffleFilesTxt$ function to shuffle images. Once we had annotations and list of annotations, we used The ElementTree library to parse xml in order to create csv file.

Our library also provides functions to save only a part or all annotations in order to create new subdataset, which contains only specific objects annotated. In our case the selected objects are Pedestrians. This task can be done via $convertAnnotations$ function for KITTI-like datasets or $createSubVOC$ function for Pascal VOC-like datasets.

Created library is open-source. Small documentation is included in source code in comments above every function.

4.3 Learning stage

Before we could start the training process, we needed to do a couple more things. First of all, we had to create training and testing record, because Object Detection API can not read just images and annotations in xml format. In order to create these files, we converted all annotations to one cvs file. In this file every row represents one object. To achieve this we used xml_to_csv function from our library. When we had these csv files, we used $generate\_tfrecord.py$ script which created required record files. Next step was modification of the pipeline configuration file of the selected model. We had to create pbtxt file with class labels – only for pedestrians in our case:

```
1 item {
2   id: 1
3   name: 'person'
4 }
```

Then we had to specify the path to this class label file in conf file that comes with the model. This file is used to control model behavior. In this file, we also had to specify the path to training and testing records. We also had to set number of classes to detect and path to class labels. You can see example below.

```
7 model {
8   ssd {
9     num_classes: 1
```

```
10
11      ...
12
13    fine_tune_checkpoint: "ssdlite_mobilenet_v2_coco_2018_05_09
        /model.ckpt"
14
15    ...
16
17  train_input_reader: {
18    tf_record_input_reader {
19      input_path: "data/train5k.record"
20    }
21    label_map_path: "annotations/obj_detection.pbtxt"
22  }
23
24    ...
25
26  eval_input_reader: {
27    tf_record_input_reader {
28      input_path: "data/test.record"
29    }
30    label_map_path: "annotations/obj_detection.pbtxt"
31    shuffle: false
32    num_readers: 1
33
34    ...
35  }
```

For Faster R-CNN model is cruial to add line *num_readers* : 1 for the right evaluation result.

When everything was ready, we started our experiment by training both Faster R-CNN and SSDLite on training set consisting of 500 the KITTI images and 800 images consisting of KITTI, CityPersons, Small Night Pedestrian Dataset and Pascal VOC. These four models were evaluated during training stage. You can see figures 4.14 and 4.15 for more detailed look on the training stage. For training we used script *models.py* from Object Detection API library. You can simply run this script by command:

```
python model_main.py
  --pipeline_config_path=training1/
      faster_rcnn_resnet101_kitti.config
  --num_train_steps=100000
```

```
--alsologtostderr
--sample_1_of_n_eval_examples=1
--model_dir=training1/
```

Where the $--pipeline\_config\_path$ serves as a path to pipeline configuration path, $--num\_train\_steps$ is optional. It can directly overwrite number of step of training. The $--model\_dir$ sets the trainig folder, where all the training data and models are stored. It is also optional, but we highly recommend to use it.

Because TensorFlow saves model every 600 seconds and only 5 newest models are saved, we created python script to periodically copy *model_dir* folder in order to save all models. Once we are done with training, we can simply select the best model from all trained models. To use this script simply run this command:

```
python periodicCopy <path to model_dir> <output path>
```

Monitoring of the training progress can be done by the TensorBoard tool which comes with TensorFlow installation. This tool creates a local web server and you can use web browser to monitor the training stage. You can run Tensorboard via command:

```
tensorboard --logdir=<path to model_dir>
```

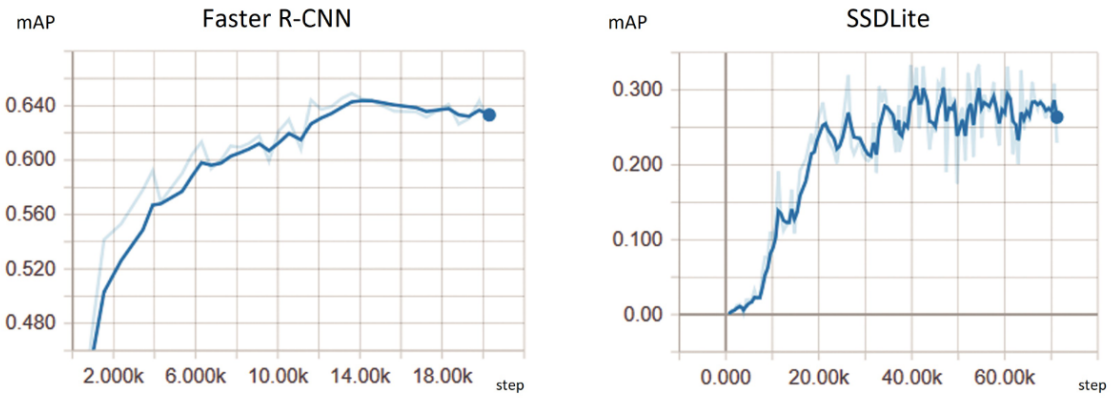


Fig. 4.14: Model evaluation during the training stage (500-frame dataset).

After these 4 models were successfully trained, we picked the best model – Faster R-CNN trained on 800 images and then we used it for pseudo-optimal model searching. The table 4.3 shows detailed look at transfer learning time. We managed to train 10 models in total. In addition to the 4 models mentioned above, we trained the Faster R-CNN on training datasets of different sizes - 10, 100, 2,000 and 5,000 images composed of all selected datasets. Then we trained the Faster R-CNN model with increased feature extractor resolution. This model is used to compare the effect of feature extractor resolution on the resulting model accuracy. The last trained model

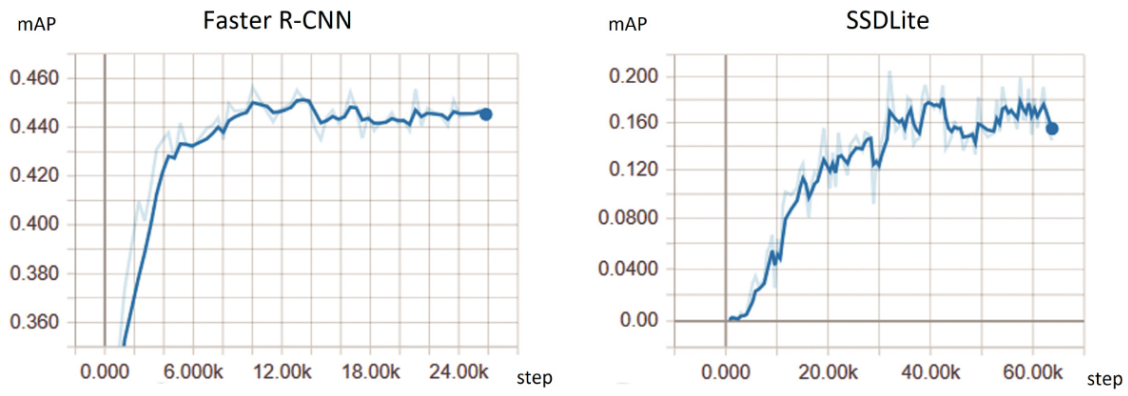


Fig. 4.15: Model evaluation during the training stage (800-frame dataset).

is SSDLite. It was trained on the same 5,000 images as Faster R-CNN, as it has proven to be the best choice. Because SSDLite has a fixed resolution of $300x300$ px, we haven't trained it to a higher resolution. See next chapter for detailed results analysis.

To deploying final version of the trained model you can use the script *export_inference_graph.py* by command:

```
python export_inference_graph.py
  --input_type image_tensor
  --pipeline_config_path= <path to pipeline config>
  --trained_checkpoint_prefix <path to model.ckpt>
  --output_directory <output folder>
```

Tab. 4.1: Training time - trained on MX 150, 2,000 and 5,000 dataset trained on Titan X.

Training Time [m]			
Faster RCNN 10	351	**SSDLite 500**	737
Faster RCNN 100	411	**SSDLite 800**	1111
Faster RCNN 500	263	**SSDLite 5k**	1692
Faster RCNN 800	378		
Faster RCNN 2k	677		
Faster RCNN 5k	4946		

4.4 Object detector

Below you can see the object detector for video stream code snippet. We had to create the TensorFlow session, load the selected model a then we can gradually take the individual camera images in the infinite loop. Each image is represented by the Numpy array so we can easily expand image dimensions to required tensor proportions. After that we created variables for boxes, scores and classes and run the detection by sess.run function. The whole script also measures detection speed and whole loop speed.

```
55 with detection_graph.as_default():
56     with tf.Session(graph=detection_graph) as sess:
57         while True:
58             start_time = time.time()
59             ret, image_np = cap.read()
60             detection_time = time.time()
61             image_np_expanded = np.expand_dims(image_np, axis
                   =0)
62             image_tensor = detection_graph.get_tensor_by_name
                   (
63                   'image_tensor:0')
64             boxes = detection_graph.get_tensor_by_name(
65                   'detection_boxes:0')
66             scores = detection_graph.get_tensor_by_name(
67                   'detection_scores:0')
68             classes = detection_graph.get_tensor_by_name(
69                   'detection_classes:0')
70             num_detections = detection_graph.
                   get_tensor_by_name(
71                    'num_detections:0')
72             (boxes, scores, classes, num_detections) = sess.
                   run(
73                    [boxes, scores, classes, num_detections],
74                    feed_dict={image_tensor: image_np_expanded})
75             detection_end_time = time.time()
```

There are 5 scripts for object detection:

```
confusion_matrix_all_in_folder.py
confusion_matrix_one_file.py
objDetection_camera.py
objDetection_picture.py
```

```
objDetection_video.py
```

The first two are for computing confusion matrix from one image / folder of images. The rest of them serve as object detectors from the web-cam video stream, single image and for video processing. All of these scripts need to be configured before use. All these files have a section for setting model parameters right after the module import part. This section is clearly marked via comment. You can set right model, path to class labels (the pbtxt file), number of classes which the model certifies, score (and IoU for confusion matrix) threshold, path to image / folder of images and so on. You can see an example from *objDetection_picture.py* below.

```
16 # Model definition
17 PATH_TO_CKPT = 'final_models/final_ADS/frozen_inference_graph
       .pb'
18 # pbtxt file with object labels
19 PATH_TO_LABELS = os.path.join('aaa_AllDataset/annotations', '
       obj_detection.pbtxt')
20 # Number of classes
21 NUM_CLASSES = 1
22 # Number of maxdetections
23 MAX_DETECTIONS = 100
24 # Detection score treshold
25 THRESHOLD = 0.7
26 # Path to input image
27 PATH_TO_IMG = "det/a.png"
```

These files can be runned by command

```
python <filename>
```

For example if you want run the *objDetection_picture.py* script, you have to run command:

```
python objDetection_picture.py
```

4.5 Automated dataset creation

Due to the difficulty of manual annotation of ground truth boxes, we decided to introduce a tool for automated creation of a dataset using a machine learning model. Since the objects are annotated using a machine learning model, the quality of annotations is strongly dependent on the model's quality. Therefore, this tool is not intended for use without a human operator and the dataset so created should always be checked.

This tool can be found in the annexes. This is the *create_dataset.py* script. Its usage is the same as for detection tools. To run:

```
python create_dataset.py
```

The tool needs to be set before use. This setting is the same as for object detection scripts. The only difference is that it also allows you to save images with visualization of marked ground truth boxes. For this option, you need to uncomment the cv2 module and its function in the code. This option is disabled by default to save computing time. During testing, a high score for the score threshold proved to be successful. Our recommended score threshold for Faster R-CNN models is 0.9.

Fig. 4.16: An example from the create_dataset script.

The script stores annotations in a format identical to the Pascal VOC annotation format. An xml file is always created for each image. The name of this file matches the image name. The entire script is designed to be easily expandable if it needs to be modified. An example from the code is shown below.

```
35 def makeDatasetRecord(detList, myFile, width, height):
36   root = ET.Element("annotation")
37   ET.SubElement(root, "filename").text = str(myFile)
38   ET.SubElement(root, "folder").text = "VOC2012"
39   ET.SubElement(root, "segmented").text = "0"
40
41   size = ET.SubElement(root, "size")
42   ET.SubElement(size, "depth").text = "3"
43   ET.SubElement(size, "width").text = str(width)
44   ET.SubElement(size, "height").text = str(height)
45
```

```
46    source = ET.SubElement(root, "source")
47    ET.SubElement(source, "annotation").text = "Dummy"
48    ET.SubElement(source, "database").text = "Dummy"
49    ET.SubElement(source, "image").text = "Dummy"
50
51    for rec in detList:
52      mobject = ET.SubElement(root, "object")
53      ET.SubElement(mobject, "name").text = str(rec[4])
54      ET.SubElement(mobject, "difficult").text = "0"
55      ET.SubElement(mobject, "occluded").text = "0"
56      ET.SubElement(mobject, "truncated").text = "0"
57      ET.SubElement(mobject, "pose").text = "Unspecified"
58
59      bndbox = ET.SubElement(mobject, "bndbox")
60      ET.SubElement(bndbox, "xmin").text = str(rec[0])
61      ET.SubElement(bndbox, "xmax").text = str(rec[2])
62      ET.SubElement(bndbox, "ymin").text = str(rec[1])
63      ET.SubElement(bndbox, "ymax").text = str(rec[3])
64
65    tree = ET.ElementTree(root)
66    tree.write(str(myFile.split('.')[0])+".xml")
```

5 EXPERIMENT RESULTS

This chapter analyzes the behavior of the trained models. These models were trained on different test datasets (see previous chapter). During training, they were evaluated by using the same test dataset. According to these results, the best model was selected for each training dataset. These resulting models were subjected to a more detailed analysis of their behavior.

5.1 Used metrics

During the training phase, only mAP was taken into account, because it is a generally accepted metric for measuring the effectiveness of models in Deep Learning. mAP means mean average precision. Tensorflow Object Detection API uses the COCO mAP metrics. This is calculated on the basis of IoU for individual detections. Tensorflow takes mAP into account for both IoU $> 50\,\%$ (this metric is also known as Pascal metric) and more complex solution which corresponds to the average AP for IoU from $50\,\%$ to $95\,\%$ with a step size of $5\,\%$.

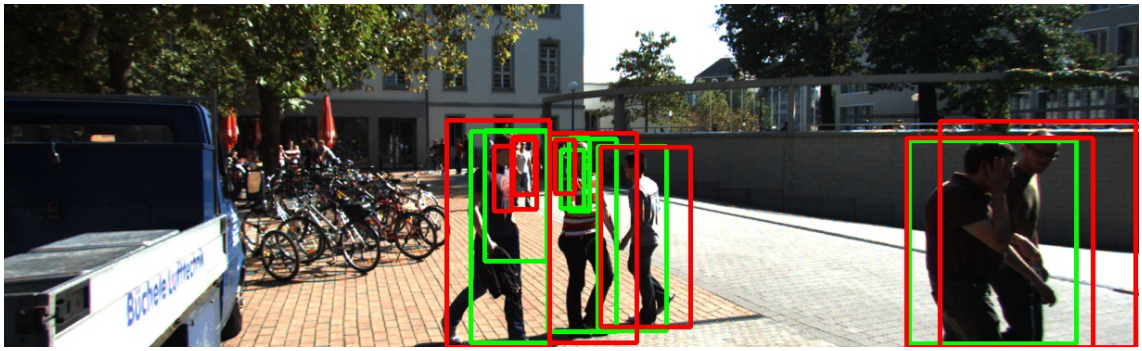


Fig. 5.1: The ground truth (red) and predicted (green) boxes for computing IoU.

IoU is an evaluation metric used to measure the accuracy of an object detector. For computing IoU we need the ground truth box and predicted box. IoU can be then computed by dividing area of overlap and area of union of these two boxes. For better understanding see figure 5.2.

If IoU is greater than some threshold we consider detection as True Positive (TP). But every ground truth box can have only one prediction box marked as TP and vice versa. This selection is based on the IoU size. Thus, the detections are divided into confusion matrix (see the table 5.1). It is a table layout that allows visualization of the performance of an machine learning model. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual (known from ground truth annotations) class or vice versa.

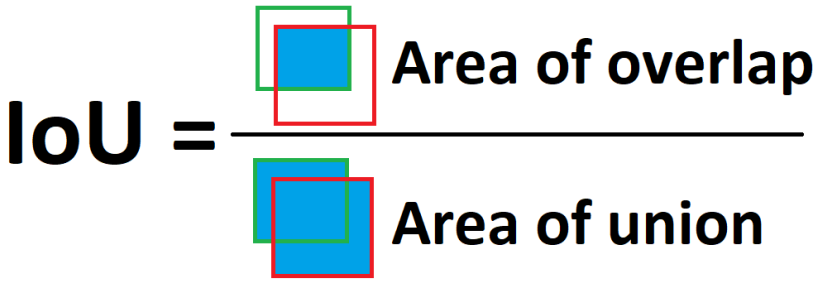


Fig. 5.2: IoU - Intersection over Union.

This matrix divides the predictions into several cases. The True Positive case means that the prediction correctly marks the object, the False Positive (FP) means that the detector detects an object but the object is not GT. On the other hand, the False Negative (FN) occurs when the detector does not detect an object. The True Negative (TN) case is set if the background is correctly marked as a background - it cannot occur for our task because the detector does not mark the background but only objects.

Tab. 5.1: Confusion matrix.

		Actual value	
		Positive	**Negative**
Predicted value	**Positive**	TP	FP
	Negative	FN	TN

Confusion matrix counts many accuracy criteria. We present the most commonly used criteria: Accuracy (5.1) - total detector accuracy , Error (5.2) - total detector error, Precision (5.3) - positive predicted value, Recall (also called Sencitivity) (5.4) - true positive rate and F1 - score (5.5) - harmonic mean of Recall and Precision values.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$error = 1 - accuracy \tag{5.2}$$

$$precision = \frac{TP}{TP + FP} \tag{5.3}$$

$$recall = \frac{TP}{TP + FN} \tag{5.4}$$

$$F1 - score = \frac{2TP}{2TP + FP + FN} \tag{5.5}$$

5.2 Detection speed of the selected methods

Another metric can be the speed of the model. We compared speed of both Faster R-CNN with a ResNet 101 as feature extractor and SSDLite with MobileNet_2 as a feature extractor. See table 5.2 for results. The MX 150 can detect detection at 1.51 FPS while applying Faster R-CNN detection algorithm. When using the Nvidia Titan X the algorithm can handle 7.52 FPS, which is an increase to 498 %. The SSDLite algorithm can handle 45.34 FPS when using the MX 150, while 47.31 FPS is achieved with Titan X. The increase is therefore only to 104.3 %. When using the MX 150 GPU the SSDLite is 30.0-times faster then Faster R-CNN, however when using Titan X GPU the SSDLite is only 6.3-times faster.

Tab. 5.2: Comparison of different meta-architectures on different NVidia GPUs.

	Detection speed [FPS]	
	MX 150	**Titan X**
Faster R-CNN	1.51	7.52
SSDLite	45.34	47.31

5.3 mAP score based on object size

As been said during the training stage we used Tensorflow Object Detection COCO metric for IoU ranging from 50 % to 95 % and IoU > 50 %. But Tensorflow Object Detection API also provides mAP based on IoU ranging from 50 % to 95 % for different object size. The objects are divided into small (max 32 x 32 px), medium (max 96 x 96 px) and large (> 96 x 96 px). See the table 5.3 for more details. Because this evaluation was performed during the training stage, it was performed only on the testing dataset. The first letter stands for used meta-architecture - F for the Faster R-CNN and S for the SSDLite. This letter is followed by a number indicating the size of the dataset. The letter H ultimately means the Faster R-CNN network with the enhanced feature extractor resolution (max dimension from 720 to 1,980 px). The @ symbol means IoU threshold. @0.5-0.95 stands for Tensorflow Object Detection API COCO metric, @0.5 stands for Pascal metric.

The results show that by increasing the number of images in the training dataset the Faster R-CNN has improved the ability to detect small objects. The Faster R-CNN trained on a 5,000-image dataset reached 29.81 % for small mAPs. Another

improvement was due to the change of the feature extractor resolution. The modified model achieved mAP on small objects of 34.89 %. On the other hand the SSDLite fails in small object detection. The best result was achieved with a dataset containing 500 images was 3.08 %.

Tab. 5.3: Comparison mAP of different models on the test dataset.

	mean Average Precision				
	All sizes @0.5-0.95	**All sizes @0.5**	**small @0.5-0.95**	**medium @0.5-0.95**	**large @0.5-0.95**
F10	0.0775	0.2481	0.0109	0.0714	0.1923
F100	0.1506	0.3794	0.0106	0.1209	0.3524
F500	0.3062	0.6453	0.0762	0.2269	0.4832
F800	0.2058	0.4518	0.0154	0.1828	0.4343
F2k	0.3027	0.5518	0.0970	0.2856	0.5418
F5k	0.5101	0.7515	0.2981	0.5327	0.6959
F5kH	0.5398	0.7639	0.3489	0.5540	0.7166
S500	0.1277	0.3321	0.0308	0.1197	0.2932
S800	0.0646	0.1928	0.0063	0.0481	0.2140
S5k	0.0990	0.2812	0.0163	0.0632	0.3091

5.4 Evaluation on validation datasets

Since a trained model can also adapt to a test model during training, a validation dataset that the model has never seen before is used for the final evaluation of accuracy. For this evaluation we used the script *eval.py* from Tensorflow Object Detection API. You can run this script via command:

```
python eval.py
  --logtostderr
  --pipeline_config_path=<path to pipeline config file>
  --checkpoint_dir=<path to model directory>
  --eval_dir=<path to eval dir>
```

The table 5.4 shows performance of the trained models. This evaluation was performed on all three validation datasets - KITTI, CityPersons and Small Night Pedestrian Dataset. Any detection with IoU threshold higher than 50 % was considered as TP.

The results confirmed the trend given in the previous chapter that by increasing the number of shots in the training dataset the precision increases. However, it is

interesting to note that for Faster R-CNN trained on 500 images, the test dataset had higher precision by 19.35 % (IoU > 50 %) than the same model trained on 800 images. After evaluating the validation dataset, it is clear that the Faster R-CNN model trained on 800 images achieved a better mAP. When comparing the results on the Small Night Pedestrian Dataset the difference is mAP 45.84 %.

However, the Faster R-CNN trained at 5,000 images again achieved the best results again, with mAP reaching 95.59 % for the KITTI dataset. Increased resolution of Faster R-CNN reached 97.03 % for KITTI. The best result of SSDLite boasts a model that also had been trained on 5,000 images. It's mAP reached 41.85 %.

It turns out that the CityPersons dataset is a challenge for networks in general. The mAP results do not exceed 40 % for Faster R-CNN and 16 % for SSDLite. So the models on this dataset do not reach even half of their success on KITTI.

Tab. 5.4: Comparison mAP of different models on the validation datasets.

	mAP - mean Average Precision		
	KITTI	**CityPersons**	**SNPD**
F10	0.3851	0.1265	0.3272
F100	0.4842	0.2818	0.5864
F500	0.6192	0.2145	0.2487
F800	0.6154	0.2714	0.7071
F2k	0.6862	0.2725	0.6944
F5k	0.9559	0.3699	0.7617
F5kH	0.9703	0.3909	0.7754
S500	0.2826	0.0246	0.1122
S800	0.3172	0.0858	0.3660
S5k	0.4185	0.1599	0.3730

5.5 Confusion matrix

Although mAP is often the only metric used in the object detection task, it is a good idea to deeply analyze the model. For this reason, we have created two scripts to evaluate the model using confusion matrix. Scripts are *confusion_matrix_one_file.py* and *confusion_matrix_all_in_folder.py*. Both of these scripts can be used with the command:

```
python confusion_matrix_one_file.py
python confusion_matrix_all_in_folder.py
```

Scripts need to be modified before use. Editing scripts is the same as for scripts designed for detection (see the chapter above). The advantage of these scripts is the ability to set both the IoU threshold and the score threshold.

To further analyze the behavior of the models, we decided to use the score threshold set to 0.7, which is the value at which the model is sure to evaluate that the object is in the given location. The reason for this setting was that if the model was to identify every detection, it would probably achieve a large number of FP detections. This would result in a great reduction in transport efficiency, since the vehicle driven by this model would often brake / stop even though it had a free path. On the contrary, we are interested in TP and FN detection for transport safety, as these are important for pedestrian safety and are more important for human life. We decided to set the threshold for IoU at 0.5.

Table 5.5 captures the confusion matrix for SSDLite trained at 500 frames. Conversely, the table 5.6 captures SSDLite trained on a dataset containing 5,000 images. It is evident that by extending the training dataset, the number of FP and FN was reduced.

Tab. 5.5: Confusion matrix for all validation datasets - SSDLite on 500 images.

		Actual value	
		Positive	**Negative**
Predicted value	**Positive**	1112	909
	Negative	5200	0

Tab. 5.6: Confusion matrix for all validation datasets - SSDLite on 5,000 images.

		Actual value	
		Positive	**Negative**
Predicted value	**Positive**	1411	763
	Negative	4901	0

The table 5.7 shows different metrics computed from confusin matrix with setting of the IoU threshold on 0.5 and the score threshold on 0.7. This table shows the results of all three validation datasets. The results show that the Faster R-CNN network trained on 5,000 images achieved the best results again. One interesting thing is the result of the precision dependence on the number of frames in the training dataset. A 10-frame dataset is 10.5 % better than a 100-frame dataset. At the same time, it is surprising that the precision dependence on the number of training images on our measurements was linear as show figure 5.3. Figures 5.4 and 5.5 show further analyse of this dependency for each of validation datasets.The dependency

was linear for Faster R-CNN except for the CityPersons dataset. This dependency was logarithmic for SSDLite and Faster RCNN in the CityPersons dataset. We justify this difference by transfer learning. The Faster R-CNN was taught on a KITTI dataset,while SSDLite was taught on a COCO dataset that is different from KITTI. This is confirmed by the result of Faster R-CNN on the CityPersons dataset, which is significantly different from the KITTI dataset.

All data measured by us can be found in the appendices to this document.

Tab. 5.7: Evaluation by our confusion matrix on all validation datasets.

	accuracy	**precison**	**recall**	**F1-score**
F10	0.2198	0.6167	0.2546	0.3604
F100	0.2688	0.5115	0.3617	0.4238
F500	0.3261	0.6781	0.3858	0.4918
F800	0.3276	0.6525	0.3969	0.4935
F2k	0.3752	0.7099	0.4431	0.5456
F5k	0.5555	0.8211	0.6320	0.7142
F5kH	0.5729	0.8098	0.6619	0.7284
S500	0.1540	0.5502	0.1762	0.2669
S800	0.1422	0.5971	0.1573	0.2490
S5k	0.1994	0.6490	0.2235	0.3325

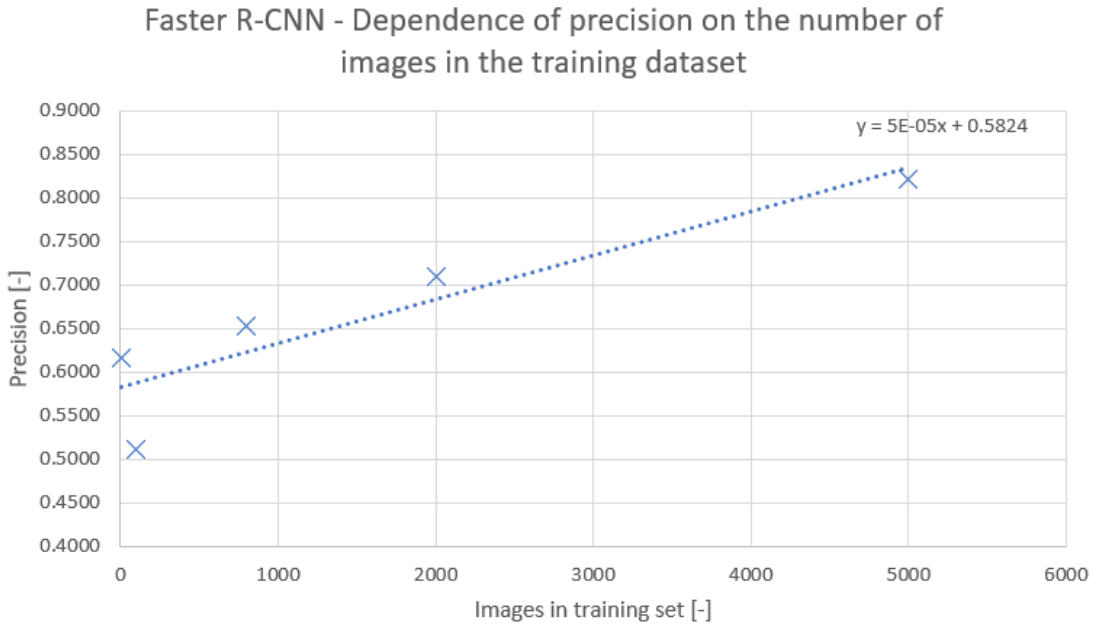


Fig. 5.3: Precision dependence on the number of training images.

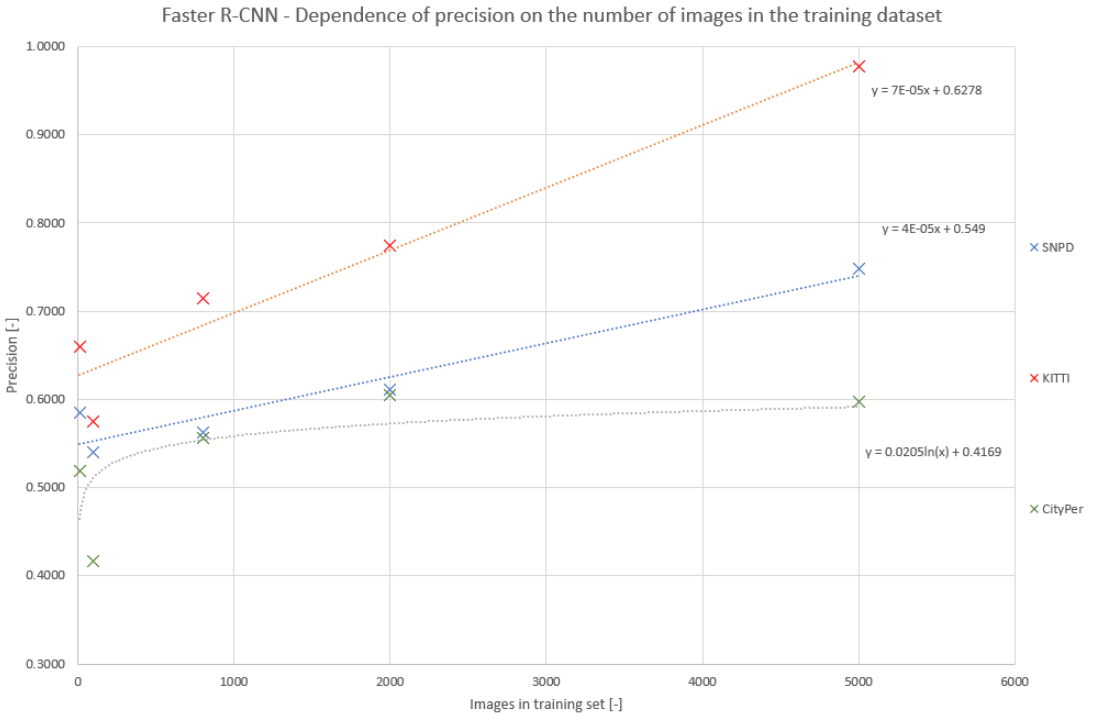


Fig. 5.4: Precision dependence on the number of training images - Faster R-CNN.

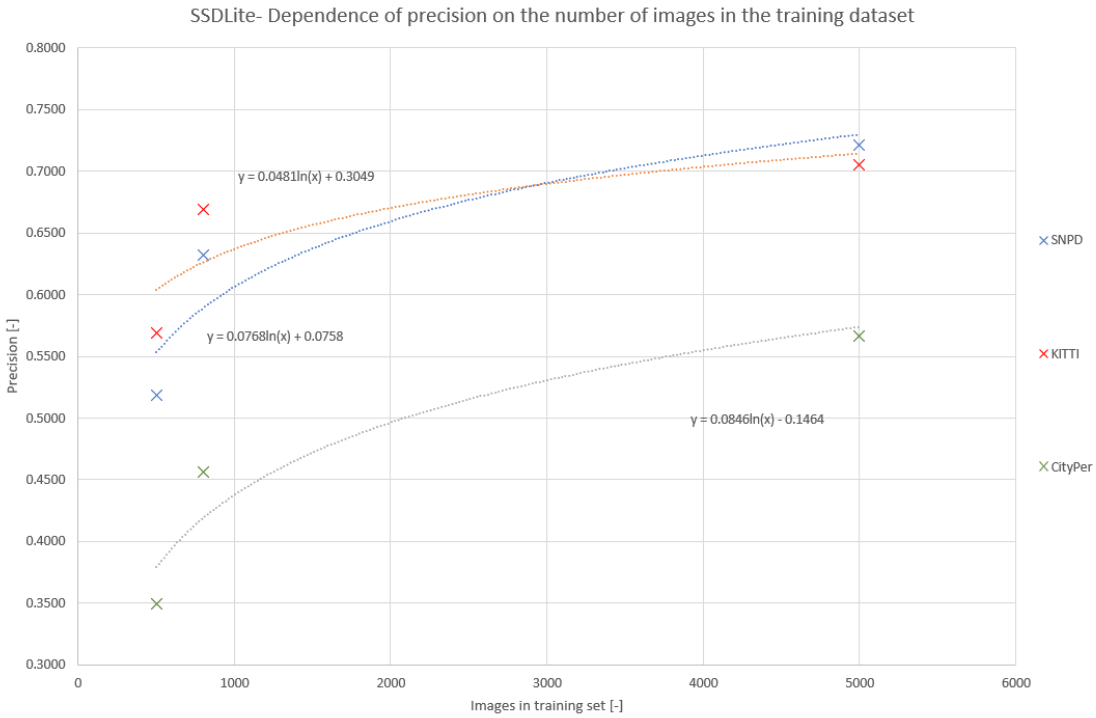


Fig. 5.5: Precision dependence on the number of training images - SSDLite.

5.6 Results discussion

In the previous chapters we discussed two meta-architectures. Faster R-CNN, which is one of the most accurate machine learning methods for image object detection and SSDLite, a derivative of the classic SSD. We have confirmed that Faster R-CNN is significantly more accurate than SSDLite. The best mAP result achieved by Faster R-CNN is 99.23 %, scored on the KITTI dataset. The best result for SSDLite is 77.09 % on the Small Night Pedestrian Dataset. However, real accuracy is lower due to environmental variability. The disadvantage of Faster R-CNN is its speed. This was only 7.52 FPS for the NVIDIA Titan X GPU. In contrast, the SSDLite speed reached 47.31 FPS on the same graphics card. The Faster R-CNN is significantly slower when using the MX 150 graphics card, while the SSDLite performance is not so bad. While using NVIDIA Titan X and Faster R-CNN the speed was 4.98-times better then while using MX 150, but while using NVIDIA Titan X and SSDLite the speed was only 1.04-times better than while using MX 150. On the other hand it can be seen that SSDLite is optimized for mobile and embedded devices. Figure 5.6 shows the behavior of the SSDLite model on a dataset containing 5,000 images. FN detections on the left are visible here. In the middle of the road you can see FP detections caused by the car on the tram tracks. On the right are the TP detections of the pedestrian on the sidewalk.

Fig. 5.6: SSDLite trained on 5,000 images.

Furthermore, we showed that by extending the training dataset, mAP is growing. This dependence was logarithmic as expected when using SSDLite models. However, the Faster R-CNN models have this dependency linear. Only when we evaluated Faster R-CNN on the CityPersons dataset, dependency was logarithmic. This can be explained by transfer learning. Pre-training of the model on the KITTI dataset caused the skip of a strongly non-linear part of the logarithmic graph. This effect was not reflected in the CityPersons dataset due to its different structure. Especially

with SSDLite, by expanding the training dataset, a large number of FP detections have been reduced. By adding Pascal VOC dataset, a great degree of generalization has been achieved. For example, models were able to detect humans only from a visible hand. This feature is generally correct, but could be undesired in some cases.

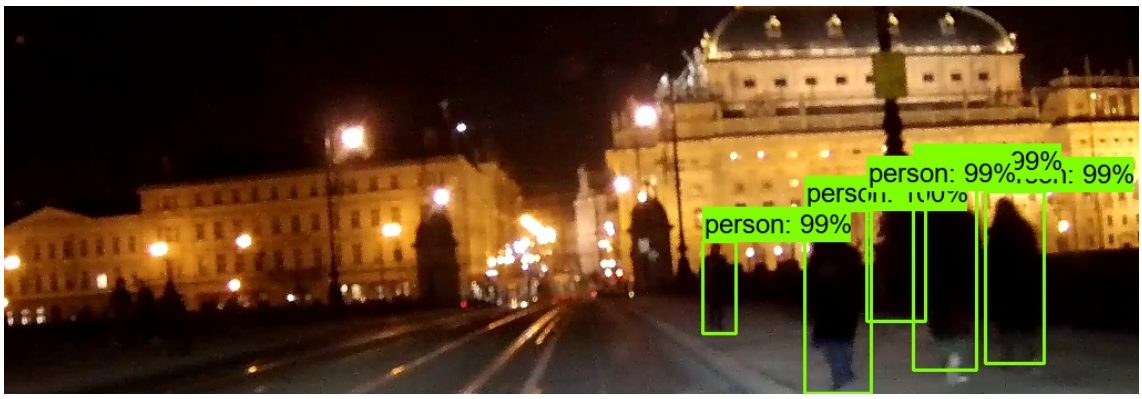


Fig. 5.7: Faster R-CNN trained on 5,000-frame dataset performing on image from the Small Night Pedestrian Dataset.

If we compare behavior of the models trained on the 500-frame dataset (KITTI only) and behavior of the models trained on the 800-frame dataset (mixed), we can see that the Faster R-CNN did not always achieve better results. Unlike SSDLite, where there was a significant improvement. This contradicts the previous statement. This anomaly may be due to the fact that the 800-frame dataset is more diverse and the network has not been properly generalized. Another option explaining this behavior is the different resolution of the extractor feature of both architectures. This behavior should be object for another research. At the same time it turns out that the addition of images from the night traffic significantly increased the success of the Small Night Pedestrian Dataset.

All our trained models have a low mAP rate on small objects. This is not surprising. In case of Faster R-CNN, this problem was solved by increasing the feature extractor resolution. This solution however places increased demands on computing power. In addition, SSDLite has a problem with people detection in a crowd. This limitation comes directly from architecture. As a suitable solution for this problem is adding a special class for the crowd. Then instead of detecting individual pedestrians we can detect pedestrian clusters. Figure 5.8 shows a comparison of the behavior of models on a crowd photograph.

Fig. 5.8: Faster R-CNN (top) and SSDLite (bottom) both trained on 5,000 images.

It is also noteworthy that both networks managed to eliminate the detection of cyclists as pedestrians. However, cyclists and motorbikers are one of the causes of FP detection. This problem should be removed by adding classes for cyclists and motorbikers.

For real-life use of these machine learning models, we recommend using Faster R-CNN supplemented with some of the object tracking algorithms, as Faster R-CNN does not reach the required speed, but it has the best accuracy compared to the meta meta-architectures. Just using one of the object tracking algorithms could solve the speed problem because some algorithms exceed 360 FPS [46].

So it turns out that diversity of the dataset plays a vital role in pedestrian detection and the more data we have for training, the better. However, to use the machine learning model in an environment that does not change and so the detected objects does not, a dataset of approximately 500 images would suffice.

6 CONCLUSION

This work deals with the detection of pedestrians in the traffic environment using machine learning techniques, namely using convolutional neural networks. Detection of persons or pedestrians is performed from the perspective of an autonomous vehicle. The use of machine learning techniques is generally beneficial for the detection of pedestrians and people in this case because it is a complex problem as both the environment and objects of interest change. At the same time we cannot make a simple compromise between accuracy and speed, because even one False Positive detection could have tragic consequences.

First, it was necessary to conduct a literature review on this topic. The first part of this work is devoted to the principle of the function of convolutional neural networks and thus creates the theoretical basis of the whole work. The second chapter, which builds on the findings mentioned in the first chapter, deals with the current State-of-the-Art solutions of convolutional neural networks for classification task only. But it also deals with meta-architectures for object detection task. These meta-architectures use the aforementioned networks as feature extractors.

As this is a supervised learning, the next chapter is dedicated to datasets suitable for pedestrian detection. The quality of the dataset used in learning directly affects the quality of the resulting model, so it is critical to pay great attention to the dataset. This chapter describes publicly available datasets. However, due to the absence of a dataset containing night-time pedestrians, we decided to create a new dataset aimed at pedestrians in the traffic environment in low light conditions like early morning, evening and night. During the creation of this dataset, we tried to encompass as many different environments as possible from the city center through villages to mountain roads. We also created the dataset under various weather conditions. Then we used this dataset together with selected datasets from this chapter for our experiment.

Based on a literature research, we selected two meta-architectures for closer testing, namely Faster R-CNN with ResNet 101 as a feature extractor and SSDLite with MobileNet_v2 as a feature extractor. The R-CNN Faster was chosen especially for its accuracy, while SSDLite was chosen mainly for its speed and because it is optimized for mobile devices. It is this feature that makes it an ideal candidate for autonomous vehicles. The Faster R-CNN architecture was pre-trained on the KITTI dataset, while SSDLite was pre-trained on the COCO dataset. As a platform for creating machine learning models, we decided to use TensorFlow and the TensorFlow Object Detection API.

The implementation of selected meta-architectures is of interest to chapter four. In addition to the implementation, we also deal with the appropriate assembly of

a training dataset from various publicly available. We decided to use the HoldOut method with a training, test and validation datasets. The size of our training dataset is 5,000 images. Dataset consists of datasets KITTI, CityPersons, Pascal VOC and our Small Night Pedestrian Dataset. To create and work with a dataset, we have created a Python function library that is also the focus of this chapter. Since the hand annotation of the dataset is very time-consuming, we have decided to create a tool that is able to automatically annotate images and thus facilitate the work of creating new datasets using a machine learning models. Last but not least, the chapter deals with created scripts for detection, whether from static image or video.

The experiment itself consists in searching for a pseudo-optimal setting for training these machine learning models. First we trained both meta-architectures on pictures only from the KITTI dataset, then on the images from different datasets. We also changed the number of training pictures for the dataset, but only for the Faster R-CNN architecture. In the last step we tried to change the resolution of the extractor feature. We then applied the best settings to the SSDLite architecture to see if there would be any improvement with another architecture.

The last chapter of this work is devoted to the analysis of results. We have confirmed that Faster R-CNN is slower but more accurate, while SSDLite has a particular problem with pedestrian groups and generally has significantly lower accuracy. However, it also achieves high speed on weaker devices. Expansion of the training dataset increased the logarithmic accuracy of SSDLite and linerage in Faster R-CNN. The diversity of the dataset also had a big impact. Especially with SSDLite, there has been a significant reduction in False Positive detection. The dataset used in the training also has a fundamental impact, with the Faster R-CNN network, the best mAP result on the KITTI dataset was 99.23 %, but significantly worse results were achieved in the evaluation on other datasets. SSDLite's best mAP result was achieved on Small Night Pedestrian Dataset with 72.09 %.

In addition to the large dataset and dataset tools we have trained ten machine learning models. Seven Faster R-CNN models and three SSDLite models. All of these models were evaluated on the CityPersons, KITTI and Small Night Pedestrian Datasets by various metrics. At the same time, we monitored the speed of detection of individual models. The assignment was thus fulfilled at all points.

Bibliography

[1] MO, Z.G. Deep learning for subway pedestrian forecast based on node camera. *Advances in Transportation Studies* [online]. Aracne Ed, 2017, **2**, 31-44 [cit. 2019-01-23]. DOI: 10.4399/97888255070584. ISSN 18245463. Available: http://www.atsinternationaljournal.com/index.php/2017-issues/special-issue-2017-vol2/922-deep-learning-for-subway-pedestrian-forecast-based-on-node-camera

[2] TOMÈ, D., F. MONTI, L. BAROFFIO, L. BONDI, M. TAGLIASACCHI and S. TUBARO. Deep Convolutional Neural Networks for pedestrian detection. *Signal Processing: Image Communication* [online]. Elsevier B.V, 2016, **47**(C), 482-489 [cit. 2019-01-23]. DOI: 10.1016/j.image.2016.05.007. ISSN 0923-5965. Available: https://arxiv.org/abs/1510.03608

[3] YANG, Dongming, Jiguang ZHANG, Shibiao XU, Shuiying GE, G. Hemantha KUMAR and Xiaopeng ZHANG. Real-time pedestrian detection via hierarchical convolutional feature. *Multimedia Tools and Applications* [online]. New York: Springer US, 2018, **77**(19), 25841-25860 [cit. 2019-01-23]. DOI: 10.1007/s11042-018-5819-6. ISSN 1380-7501. Available: https://link.springer.com/article/10.1007%2Fs11042-018-5819-6

[4] ZHANG, Liliang, Liang LIN and Xiaodan LIANG. Is faster R-CNN doing well for pedestrian detection?. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* [online]. Springer Verlag, 2016, **9906**, 443-457 [cit. 2019-01-23]. DOI: 10.1007/978-3-319-46475-6_28. ISBN 9783319464749. ISSN 03029743. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-46475-6_28

[5] HUANG, Jonathan, Vivek RATHOD, Chen SUN, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *ArXiv.org* [online]. 2017, **2017**, 21 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1611.10012

[6] KIM, Chloe Eunhyang, Mahdi Maktab Dar OGHAZ, Jiri FAJTL, Vasileios ARGYRIOU and Paolo REMAGNINO. *A Comparison of Embedded Deep Learning Methods for Person Detection* [online]. 2018, **2019**, 8 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1812.03451

[7] TALPAERT, Victor, Ibrahim SOBH, B Ravi KIRAN, Patrick MANNION, Senthil YOGAMANI, Ahmad EL-SALLAB and Patrick PEREZ.

Exploring applications of deep reinforcement learningfor real-world autonomous driving systems [online]. 2019, **2019**, 9 [cit. 2019-04-25]. Available: https://research.thea.ie/handle/20.500.12065/2400

[8] CHE, Chenyi, Ari SEFF, Alain KORNHAUSER and Jianxiong XIAO. *DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving* [online]. 2015, , 9 [cit. 2019-04-25]. Available: http://openaccess.thecvf.com/content_iccv_2015/html/Chen_DeepDriving_Learning_Affordance_ICCV_2015_paper.html

[9] DJURIC, Nemanja, Vladan RADOSAVLJEVIC, Henggang CUI, Thi NGUYEN, Fang-Chieh CHOU, Tsung-Han LIN and Jeff SCHNEIDER. *Short-term Motion Prediction of Traffic Actors for Autonomous Driving using Deep Convolutional Networks* [online]. 2018, , 7 [cit. 2019-04-25]. Available: https://arxiv.org/abs/1808.05819

[10] WU, Bichen, Alvin WAN, Forrest IANDOLA, Peter h. JIN and Kurt KEUTZER. *SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving* [online]. 2016 [cit. 2019-04-25]. Available: https://arxiv.org/abs/1612.01051

[11] KRIZHEVSKY, Alex, Ilya SUTSKEVE and Geoffrey E. HINTON. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* [online]. Association for Computing Machinery, 2017, **60**(6), 84-90 [cit. 2019-01-23]. DOI: 10.1145/3065386. ISSN 00010782. Available: https://dl-acm-org.ezproxy.lib.vutbr.cz/citation.cfm?doid=3098997.3065386

[12] SIMONYAN, Karen and Andrew ZISSERMAN. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Arxiv.org* [online]. 2014, , 14 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1409.1556

[13] VGG16. In: *Heuritech Le Blog* [online]. France: Heuritech Le Blog, 2016, 29 février 2016 [cit. 2019-01-23]. Available: https://heuritech.files.wordpress.com/2016/02/vgg16.png?w=768

[14] SZEGEDY, Christian, Wei LIU, Yangqing JIA, et al. Going Deeper with Convolutions. *Arxiv.org* [online]. , 12 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1409.4842

[15] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN and Jian SUN. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*

[online]. IEEE Computer Society, 2016, **2016-**, 770-778 [cit. 2019-01-23]. ISBN 9781467388511. ISSN 10636919. Available: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/xpl/mostRecentIssue.jsp?punumber=7776647&punumber=7776647

[16] GIRSHICK, Ross, Jeff DONAHUE, Trevor DARRELL and Jitendra MALIK. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* [online]. New York: IEEE, 2014, s. 580-587 [cit. 2019-01-23]. DOI: 10.1109/CVPR.2014.81. Available: https://arxiv.org/abs/1311.2524

[17] GIRSHICK, Ross. Fast R-CNN. *Arxiv.org* [online]. 2015, , 9 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1504.08083

[18] REN, Shaoqing, Kaiming HE, Ross GIRSHICK and Jian SUN. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. IEEE Computer Society, 2017, **39**(6), 1137-1149 [cit. 2019-01-23]. DOI: 10.1109/TPAMI.2016.2577031. ISSN 01628828. Available: https://arxiv.org/abs/1506.01497

[19] DAI, Jifeng, Yi LI, Kaiming HE and Jian SUN. R-FCN: Object Detection via Region-based Fully Convolutional Networks. *Arxiv.org* [online]. 2016, , 11 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1605.06409

[20] HE, Kaiming, Georgia GKIOXARI, Piotr DOLLÁR and Ross GIRSHICK. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. IEEE Computer Society, 2018, , xocs:firstpage xmlns:xocs=""/ [cit. 2019-01-23]. DOI: 10.1109/TPAMI.2018.2844175. ISSN 01628828. Available: https://arxiv.org/pdf/1703.06870.pdf

[21] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU and Alexander C. BERG. SSD: Single Shot MultiBox Detector. *Arxiv.org* [online]. , 17 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1512.02325

[22] REDMON, Joseph, Santosh DIVVALA, Ross GIRSHICK and Ali FARHADI. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* [online]. Las Vegas: IEEE Computer Society, 2016, **2016-**, s. 779-788 [cit. 2019-01-23]. ISBN 9781467388511. ISSN 10636919. Available: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/7780460

[23] REDMON, Joseph and Ali FARHADI. YOLO9000: Better, Faster, Stronger. *Arxiv.org* [online]. 2016, , 9 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1612.08242

[24] REDMON, Joseph and Ali FARHADI. YOLOv3: An Incremental Improvement. *Arxiv.org* [online]. 2018, , 6 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1804.02767

[25] LIN, Tsung-Yi, Priya GOYAL, Ross GIRSHICK, Kaiming HE and Piotr DOLLÁR. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. IEEE Computer Society, 2018, , 10 [cit. 2019-01-23]. DOI: 10.1109/TPAMI.2018.2858826. ISSN 01628828. Available: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/8417976

[26] Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). *Medium.com* [online]. medium.com, 2018 [cit. 2019-01-23]. Available: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

[27] LAN, Wenbo, Jianwu DANG, Yangping WANG and Song WANG. Pedestrian Detection Based on YOLO Network Model. In: *Mechatronics and Automation (ICMA), 2018 IEEE International Conference on* [online]. Changchun, China: IEEE, 2018, s. 1547-1551 [cit. 2019-01-23]. DOI: 10.1109/ICMA.2018.8484698. ISBN 9781538660744. Available: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/8484698

[28] LIN, Tsung-Yi, Michael MAIRE, Serge BELONGIE, et al. Microsoft COCO: Common Objects in Context. *Arxiv.org* [online]. 2014, , 15 [cit. 2019-01-23]. Available: https://arxiv.org/abs/1405.0312

[29] *COCO: Comon Objects in Context* [online]. Cornell University: Cornell University, Microsoft, 2014 [cit. 2019-01-23]. Available: http://cocodataset.org/#home

[30] GEIGER, A, P LENZ and R URTASUN. Are we ready for autonomous driving? The KITTI vision benchmark suite. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* [online]. Chicago: IEEE, 2012, s. 3354-3361 [cit. 2019-03-14]. DOI: 10.1109/CVPR.2012.6248074. ISBN 9781467312264.

[31] *KITTI Vision Benchmark Suite* [online]. Chicago: Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, 2012 [cit. 2019-01-23]. Available: http://www.cvlibs.net/datasets/kitti/

[32] *Caltech Pedestrian Detection Benchmark* [online]. California: California Institute of Technology, 2009 [cit. 2019-01-23]. Available: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/

[33] *INRIA Person Dataset* [online]. Grenoble, Francie: Grenoble Institute of Technology, 2005 [cit. 2019-01-23]. Available: http://pascal.inrialpes.fr/data/human/

[34] *Penn-Fudan Database for Pedestrian Detection and Segmentation* [online]. Pennsylvania: Penn Engineering, 2007 [cit. 2019-01-23]. Available: https://www.cis.upenn.edu/ jshi/ped_html/

[35] *The PASCAL Visual Object Classes Homepage* [online]. United Kingdom: Pascal VOC, 2005 [cit. 2019-01-23]. Available: http://host.robots.ox.ac.uk/pascal/VOC/

[36] *City Shapes Dataset: Semantic Understanding of Urban Street Scenes* [online]. Germany: City Shapes, 2016 [cit. 2019-01-23]. Available: https://www.cityscapes-dataset.com/

[37] *ETH Zurich: Datasets* [online]. Zurich: ETH Computer Vison Lab, 2003 [cit. 2019-01-23]. Available: http://www.vision.ee.ethz.ch/en/datasets/

[38] *Robust Multi-Person Tracking from Mobile Platforms* [online]. Zurich: ETHZ, 2007 [cit. 2019-01-23]. Available: https://data.vision.ee.ethz.ch/cvl/aess/dataset/

[39] *Mall Dataset: crowd counting dataset* [online]. Singapore: Nanyang Technological University, 2014 [cit. 2019-01-23]. Available: http://personal.ie.cuhk.edu.hk/ ccloy/downloads_mall_dataset.html

[40] NEUMANN, Lukáš, Michelle KARG, Shanshan ZHANG, et al. *NightOwls: A Pedestrians at Night Dataset* [online]. 2019, , 15 [cit. 2019-04-27]. Available: www.nightowls-dataset.org/

[41] TZUTALIN. LabelImg. *GitHub* [online]. [cit. 2019-04-25]. Available: https://github.com/tzutalin/labelImg

[42] *TensorFlow: An end-to-end open source machine learning platform* [online]. Google [cit. 2019-04-25]. Available: https://www.tensorflow.org/

[43] Object Detection API. *GitHub* [online]. 2017 [cit. 2019-04-25]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection

[44] NVIDIA TITAN X. In: *NVIDIA* [online]. [cit. 2019-05-05]. Available: https://www.nvidia.com/cs-cz/titan/titan-xp/

[45] Cocoapi. *GitHub* [online]. [cit. 2019-04-25]. Available: https://github.com/philferriere/cocoapi

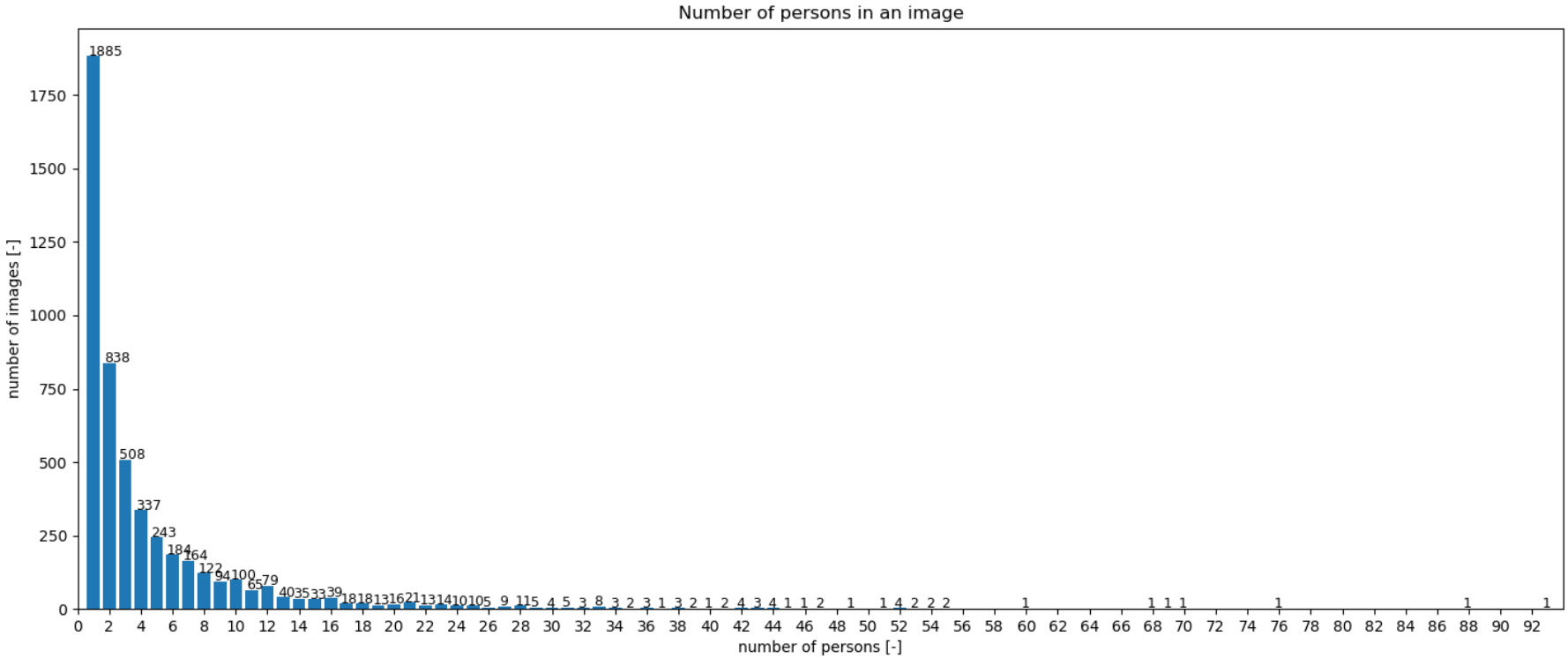
[46] WU, Yi, Jongwoo LIM a Ming-hsuan YANG. Object Tracking Benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. IEEE, 2015, **37**(9), 1834-1848 [cit. 2019-04-29]. DOI: 10.1109/TPAMI.2014.2388226. ISSN 0162-8828. Available: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/7001050

List of symbols, physical constants and abbreviations

AI	Artificial Intelligence
API	Application Program Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
cuDNN	library of primitives for deep neural networks
FN	False Negative
FP	False Positive
FPS	Frames per Second
GPS	Global Positioning System
GPU	Graphics Processing Unit
GT	Ground Truth
ICCV	International Conference on Computer Vision
ILSVRC	ImageNet Large Scale Visual Recognition Competition
IoU	Intersection over Union
mAP	mean Average Precision
NN	Neural Network
OOP	Object-oriented programming
ReLU	Rectified Linear Unit
RoI	Region of Interest
RPN	Region Proposal Network
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

List of appendices

A Histogram of pedestrian distribution in images for 5,000-frames dataset



B NVIDIA's Product comparison

Product	Compute Capability	Theoretical performance [TFLOPS]	TDP [W]	Max memory [Gb]	Bandwidth [Gb/s]
Titan X	6.1	6.7	250	12	336.6
MX 150	6.1	1.1	25	4	48.1
Tesla T4	7.4	8.1	70	16	320.0
Tesla K80	3.7	4.1	300	12	240.6
Quadro RTX 8000	7.5	16.3	260	48	672.0
Quadro P5200	6.1	8.9	100	16	230.4
Quadro K610M	3.5	0.4	30	1	20.80
NVIDIA TITAN RTX	7.5	16.3	280	24	672.0
Geforce RTX 2080 Ti	7.5	13.5	250	11	616.0
GeForce 930M	5.0	0.7	33	2	14.40

C Models evaluation

C.1 Validation datasets - divided

Faster R-CNN 10										
					val	val - confusion matrix - score 0.7+ IoU 0.5				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	55	39	108	0	0.3272	0.272277	0.727723	0.337423	0.585106	0.428016
Kitti	1170	605	1822	0	0.3851	0.325271	0.674729	0.391043	0.659155	0.490875
CityPed	382	355	2775	0	0.1265	0.10877	0.89123	0.121001	0.518318	0.196199
Faster R-CNN 100										
					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	95	81	68	0	0.5864	0.3893	0.6107	0.5828	0.5398	0.5605
Kitti	1455	1073	1537	0	0.4842	0.3579	0.6421	0.4863	0.5756	0.5272
CityPed	733	1026	2424	0	0.2818	0.1752	0.8248	0.2322	0.4167	0.2982
Faster R-CNN 500										
					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	94	53	69	0	0.2487	0.4352	0.5648	0.5767	0.6395	0.6065
Kitti	1772	868	1220	0	0.6192	0.4591	0.5409	0.5922	0.6712	0.6293
CityPed	569	235	2588	0	0.2145	0.1677	0.8323	0.1802	0.7077	0.2873
Faster R-CNN 800										
					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	117	91	46	0	0.7071	0.4606	0.5394	0.7178	0.5625	0.6307
Kitti	1664	665	1328	0	0.6154	0.4550	0.5450	0.5561	0.7145	0.6254
CityPed	724	578	2433	0	0.2714	0.1938	0.8062	0.2293	0.5561	0.3247
Faster R-CNN 2k										
					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	113	72	50	0	0.6944	0.4809	0.5191	0.6933	0.6108	0.6494
Kitti	1892	553	1100	0	0.6862	0.5337	0.4663	0.6324	0.7738	0.6960
CityPed	792	518	2365	0	0.2725	0.2155	0.7845	0.2509	0.6046	0.3546

Faster R-CNN 5k

					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	119	40	44	0	0.7617	0.5862	0.4138	0.7301	0.7484	0.7391
Kitti	2733	64	259	0	0.9559	0.8943	0.1057	0.9134	0.9771	0.9442
CityPed	1137	765	2020	0	0.3699	0.2899	0.7101	0.3602	0.5978	0.4495

Faster R-CNN 5k HD

					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	131	83	32	0	0.7754	0.5325	0.4675	0.8037	0.6121	0.6950
Kitti	2834	22	158	0	0.9703	0.9403	0.0597	0.9472	0.9923	0.9692
CityPed	1213	876	1944	0	0.3909	0.3008	0.6992	0.3842	0.5807	0.4624

SSDLite 500

					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	14	13	149	0	0.1122	0.0795	0.9205	0.0859	0.5185	0.1474
Kitti	1039	786	1953	0	0.2826	0.2750	0.7250	0.3473	0.5693	0.4314
CityPed	59	110	3098	0	0.0246	0.0181	0.9819	0.0187	0.3491	0.0355

SSDLite 800

					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	43	25	120	0	0.3660	0.2287	0.7713	0.2638	0.6324	0.3723
Kitti	698	345	2294	0	0.3172	0.2092	0.7908	0.2333	0.6692	0.3460
CityPed	252	300	2905	0	0.0858	0.0729	0.9271	0.0798	0.4565	0.1359

SSDLite 5k

					val	val - confusion matrix				
	TP	FP	FN	TN	mAP@0.5	accuracy	error	recall	precision	F1-score
SNPD	31	12	132	0	0.3730	0.1771	0.8229	0.1902	0.7209	0.3010
Kitti	879	367	2113	0	0.4185	0.2617	0.7383	0.2938	0.7055	0.4148
CityPed	501	384	2656	0	0.1599	0.1415	0.8585	0.1587	0.5661	0.2479

C.2 Validation datasets - united

All validation datasets	val - confusion matrix - score 0.7+ IoU 0.5				
	accuracy	error	recall	precision	F1-score
Faster R-CNN 10	0.2198	0.7802	0.2546	0.6167	0.3604
Faster R-CNN 100	0.2688	0.7312	0.3617	0.5115	0.4238
Faster R-CNN 500	0.3261	0.6739	0.3858	0.6781	0.4918
Faster R-CNN 800	0.3276	0.6724	0.3969	0.6525	0.4935
Faster R-CNN 2,000	0.3752	0.6248	0.4431	0.7099	0.5456
Faster R-CNN 5,000	0.5555	0.4445	0.6320	0.8211	0.7142
Faster R-CNN 5,000 HD	0.5729	0.4271	0.6619	0.8098	0.7284
SSDLite 500	0.1540	0.8460	0.1762	0.5502	0.2669
SSDLite 800	0.1422	0.8578	0.1573	0.5971	0.2490
SSDLite 5,000	0.1994	0.8006	0.2235	0.6490	0.3325

D Media content

- / — Media root
  - SNPD — Small Night Pedestrian Dataset
    - train — Training part
    - test — Testing part
  - Tensorflow — Tensorflow workspace
    - Code Corections — Code corections for Windows 10
      - detection_inference.py
      - object_detection_evaluation.py
      - tf_example_parser.py
    - final_models — Models files
      - final_ADF2k — Faster R-CNN trained on 2k-frames dataset
      - final_ADF5k — Faster R-CNN trained on 5k-frames dataset
      - final_ADF10 — Faster R-CNN trained on 10-frames dataset
      - final_ADF100 — Faster R-CNN trained on 100-frames dataset
      - final_ADF800 — Faster R-CNN trained on 800-frames dataset
      - final_ADFR5k — Faster R-CNN HD trained on 5k-frames dataset
      - final_ADS5k — SSDLite trained on 5k-frames dataset
      - final_ADS800 — SSDLite trained on 800-frames dataset
      - final_KOF500 — Faster R-CNN trained on 500-frames dataset
      - final_KOS500 — SSDLite trained on 500-frames dataset
    - scripts — Created scripts
      - confusion_matrix_all_in_folder.py
      - confusion_matrix_one_file.py
      - create_dataset.py
      - dataset_info.py
      - dataset_tools.py
      - objDetection_camera.py
      - objDetection_picture.py
      - objDetection_video.py
      - periodicCopy.py
    - workspace — Files for datasets
      - evalCityPed_labels.csv — CSV file for creating record file
      - evalKitti_labels.csv — CSV file for creating record file
      - evalSNPD_labels.csv — CSV file for creating record file
      - obj_detection.pbtxt — Class label file
      - test_labels.csv — CSV file for creating record file
      - train2k_labels.csv — CSV file for creating record file
      - train5k_labels.csv — CSV file for creating record file
      - train10_labels.csv — CSV file for creating record file
      - train100_labels.csv — CSV file for creating record file
      - train500_labels.csv — CSV file for creating record file
      - train800_labels.csv — CSV file for creating record file
  - xtilgn00.pdf — Master's thesis document